BOSTON UNIVERSITY
COLLEGE OF ENGINEERING

Thesis

SPEECH EVENT DETECTION USING STRICTLY TEMPORAL INFORMATION

by

ARIEL SALOMON

S.B., Massachusetts Institute of Technology, 1996

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
2000

Approved by:

First Reader: _____

Dr. Carol Espy-Wilson, Associate Professor,
Department of Electrical and Computer Engineering,
Boston University

Second Reader: _____

Dr. Laurel Carney, Associate Professor,
Department of Biomedical Engineering,
Boston University

Third Reader: _____

Dr. W. C. Karl, Assistant Professor,
Department of Electrical and Computer Engineering,
Boston University

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Carol Espy-Wilson for providing direction and support in pursuing this research, and helping me to direct my course of study at Boston University. I would also like to thank my parents Michael and Alyza Lee Salomon for encouraging me to reach this point in my studies and my life, providing moral support, as well as for passing along the wisdom that "the most important quality of a thesis is that it's *done*".

This work was supported by NSF grant #SBR-9729688.

SPEECH EVENT DETECTION
USING STRICTLY TEMPORAL INFORMATION

ARIEL SALOMON

Boston University, College of Engineering, 2000

Major Professor: Carol Espy-Wilson Associate Professor of Electrical Engineering

ABSTRACT

A major problem in the development of speech recognition systems is the understanding of speech in noise, or from reduced spectral information. The problem of speech perception in noise has a long history, and the particular spectral contributions to speech intelligibility are well understood. Recent studies show that, particularly in degraded environments, another source of information that may be of use is that of temporal cues—i.e. the temporal structure of components of the speech signal. These cues are not primarily targeted by traditional speech recognition systems, but the auditory system is sensitive to subtle temporal effects which suggests that this information is available and is used in human speech recognition.

In light of these facts, this thesis addresses the development of algorithms that specifically target temporal structure. This study analyzes how much information about important events (toward detection of linguistically-motivated landmarks) in a speech signal can be detected using a set of temporal cues. Specific parameters that have been developed include an adaptive energy difference measure for onset and offset detection, as well as periodicity (and pitch) detection to segregate periodic and aperiodic components of the speech signal.

In order to evaluate the detector, an automated system was developed which

posited events based on labels from sentences in the TIMIT corpus. The resulting comparison showed that 70.8% of expected events were detected, in particular 87.1% of the most perceptually salient events (77.2% and 89.6% on the training set), with over 90% success for particular event classes. It is expected that future improvements can be attained, especially when temporal cues are integrated with spectral information.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This thesis investigates the use of temporal information for extraction of linguistically relevant details from a speech signal, in the context of a larger project building a knowledge-based speech recognition system based on distinctive features. The larger project is the development of a structured method for analysis of speech signals based on knowledge of the nature of those signals, as understood from studies of the acoustical properties of speech production (refer to [1] for exhaustive coverage of this body of knowledge). Goals of this thesis include extending theoretical understanding of the extent and type of temporal information in a speech signal and its correspondence to linguistic details. More directly, the goal is a prototype system for extraction of temporal cues which could be used as a component of an automated speech recognition system.

The motivations for this work come from studies of speech perception. A major problem in the development of speech recognition systems is the understanding of speech in noise, or from reduced spectral information. The study of speech perception in noise has a long history, and the effects of spectral degradation (such as adding noise) on speech intelligibility are well understood, note for example the work of Miller & Nicely [2]. However, recent studies show that another source of information that may be of use, particularly in degraded environments, is from temporal cues—information derived from the temporal structure of components of the speech signal. These cues are not targeted by traditional speech recognition systems,

which generally focus on spectral features using data-derived spectral templates. As the auditory system is sensitive to subtle temporal effects, such as phase locking of auditory nerve firing to periodic signals, it suggests that this information should be available for use in human speech recognition. This is shown to be the case in studies by Van Tasell et al. [3, 4], Shannon et al. [5], and Turner et al. [6] (among others) which demonstrate the ability of human listeners to recognize speech—particularly consonant manner, nasality, and voicing—from primarily temporal cues.

The goals of this work are algorithms which can be integrated into the front end of a knowledge-based speech recognition system. The system developed in this thesis locates acoustically abrupt events in a speech signal, toward the discovery of *landmarks* in speech, or linguistically important points of analysis in the signal. Landmarks are a phonologically motivated set of locations in the signal at which further feature extraction is required, for the purpose of classifying adjacent or surrounding regions. Locating a set of landmarks in the speech signal is intended to direct an efficient analysis of linguistically-relevant information in the signal, as well as to provide a first-order analysis of the signal content.

To introduce the linguistically motivated nature of this research, background regarding the production and perception of speech, and correspondingly about the definition of landmarks as understood in this study, is presented in Section 1.1. Further discussion of the motivation for this work from research on perception of speech from temporal cues is in Section 1.2. Some previous work on the use of temporal parameters is discussed in Section 1.3. Finally, the specific goals of this work are considered in Section 1.4.

1.1 Background: Speech production and perception

In the human speech production system, a sequence of discrete elements are translated into an analog, continuous acoustic signal by the vocal apparatus [7]. The

process of understanding speech can be considered to be a reconstruction of the discrete stream of symbols (whether phonemes, words, etc.) from the speech signal. In other words, speech understanding is the process of trying to determine the set of symbols that were intended to be transmitted, based on the acoustical signal received by the auditory system[1]. To perform the task of speech recognition from the acoustic signal, knowledge of the production mechanisms must be understood: both the functional apparatus, and the ways in which it is used (particularly with respect to representations in the brain).

Speech is produced by a source (air from the lungs, forced through the glottis) exciting a complex adaptive filter (the vocal tract), which changes as the articulators (tongue, lips, etc.) move to produce different sounds. The source includes the vocal folds, which have the capability to produce voicing, a regular periodic excitation at a rate referred to as the fundamental frequency (F0). The vocal tract can also be adjusted to produce a turbulent source further forward in the vocal tract, such as for a fricative consonant (e.g. /s/). This forward source can be generated in the presence of voicing (e.g. /z/). Further, the articulators can be moved in more subtle ways to modify the vocal tract filter, most importantly characterized by the major poles of the system, referred to as formants. The filter stage produces a number of important phonetic distinctions, most importantly the movement of the body of the tongue during vocalic regions which adjusts the positions of the first two formants to produce different vowel sounds.

Through implicit knowledge of the way in which this vocal apparatus works in general, and information about the language such as the types of distinctions that are expected (the phonetic dictionary of the particular language being spoken), a machine or human listener tries to recover a discrete sequence from the signal. Use of

[1] Of course, the acoustical information will be combined with other types of information, including knowledge of the language, discourse context, and other modalities such as visual cues (e.g. lip-reading).

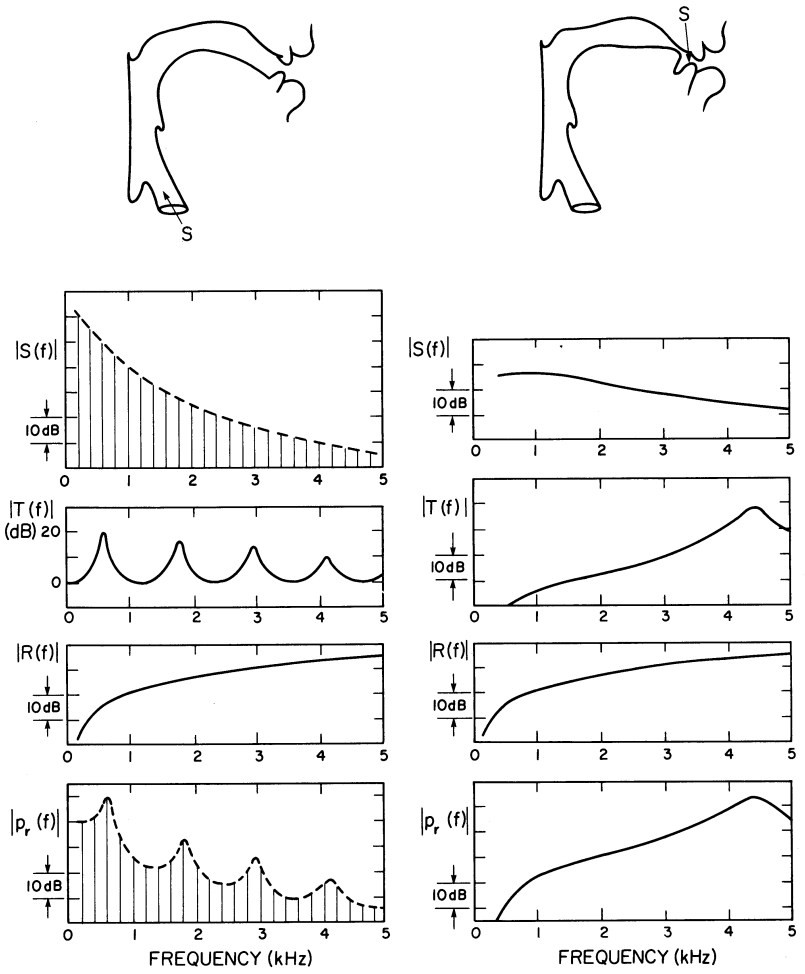

Figure 3.1 Sketches indicating components of the output spectrum $|p_r(f)|$ for a vowel and a fricative consonant. The output spectrum is the product of a source spectrum $S(f)$, a transfer function $T(f)$, and a radiation characteristic $R(f)$. The source spectra are similar to those derived in figures 2.10 and 2.33 in chapter 2. For the periodic source, $S(f)$ represents the amplitudes of spectral components; for the noise source, $S(f)$ is amplitude in a specified bandwidth. See text.

(a) (b)

Figure 1.1: Source-filter model of speech production, taken from [1]. The speech signal is produced by exciting the vocal tract with an excitation signal, either with the periodic source of the vocal folds, or by generating a turbulent source further forward in the vocal tract.

knowledge-based representations in speech recognition involves explicit application of knowledge of speech production and linguistics, and corresponding implications for speech perception, to the recognition problem.

When considering the initial analysis of the acoustic signal, an important level of representation is at the phonological level: as a sequence of *phonemes* (speech sounds). These phonemes are *segments* (which may be overlapping) that can be thought of as being made up of bundles of *features* describing how they are produced. Features can be divided into two types: manner features and place features. Manner features describe the overall mode of articulation of a speech segment, e.g. the distinction between /sh/ and /y/, though both are produced with roughly the same part of the tongue. Place features describe more specific details of the articulators used, such as differentiating the stop consonant /b/, produced by the lips against the upper teeth (labiodental) from /d/, produced with the tongue tip against the alveolar ridge. A summary of major manner classes[2] is listed in Table 1.1. An-

[2] Corresponding to the primary manner features, ±**sonorant**, ±**syllabic**, ±**nasal**, ±**continuant**.

Table 1.1: Manner classes of speech. Manner class of a phonetic segment is the mode of production, in terms of classification of the configuration of the vocal tract.

Class	Vocal tract configuration
vowels	vocal tract open, excitation is periodic
semivowels	vocal tract slightly constricted, excitation is periodic
nasals	vocal tract closed but with nasal coupling, excitation is periodic
fricatives	vocal tract narrowly constricted to produce turbulent noise, primary excitation is *aperiodic*
stops	vocal tract completely closed to build up pressure (silence) and air is suddenly released (transient), primary excitation is *aperiodic*

other important manner feature is voicing[3] (/s/ vs. /z/), which is the presence of periodic excitation from the glottal source.

1.1.1 Landmarks and irregular sampling

Not all regions of a signal have the same information content: steady-state portions can be sampled slowly to determine overall properties, while abrupt changes can contain a significant number of linguistically important cues in a concentrated region. A useful concept is to consider speech segments in terms of the vocal tract approaching a target position where the acoustical signal results in the most reliable information about the features of the segment being produced. The transition regions between these points may contain information about the segment, but most importantly in the type of transitions in and out of the target position. Landmarks can be considered to be these points where a target position has been reached (in a steady state region), and where a target results in an abrupt change in signal quality (e.g. closure or release point of a consonant).

The placement and types of landmarks are strongly related to the manner features of the corresponding segments [8]. As these locations are important places to initiate further detailed analysis of the signal, this implies an early stage of processing in a knowledge-based speech recognition system ought to be location of these landmarks. This is a similar approach to those taken by Espy-Wilson [9, 10], and Liu [11, 12]. Further analysis starting at the points where landmarks are located will enable the system to identify the features of the segments that were produced. The directed analysis in this model is intended both to decrease the overall processing load, as well as to disregard less reliable information in the signal.

[3] i.e. the feature ±**voice**.

1.1.2 Acoustic events

This study examines a method for extraction of abrupt acoustic events—locations where abrupt changes occur in a speech signal. The specific parameters used to determine these locations depend on changes in spectral energy or in the periodicity content of the signal. This work relies on the assumption that a majority of the landmarks, as well as related events which may not be landmarks in the most strictly linguistically defined sense, will be locatable as events in the speech signal. These events will be cues for the location of landmarks in the speech signal. It is proposed that the abrupt events capture the 'temporal dynamics' of the speech signal. The object of detecting these events is that they are related to a set of landmarks in the speech signal.

Identification of events is the first stage leading toward location of landmarks in the signal. This of course will be followed by higher-level composition for recognition of segments, words, and any further interpretation by higher levels of processing. These later stages of processing increasingly involve linguistic knowledge such as the set of contrastive segments available in the language, a lexicon of words, etc. However, at the level of the event detector only minimal knowledge about speech production is applied.

Sample event-level labels for an utterance are shown in Figure 1.2 in panel (3) below the corresponding TIMIT segment labels. Note in particular the event labeled $+\mathbf{c}$ at 657ms, which is the stop burst for the consonant /p/, and the following $-\mathbf{c}$ and $+\mathbf{v}$ events corresponding to the end of the initial burst and then the onset of voicing. In the case of the stop consonant, the burst (between the $+\mathbf{c}$ and $-\mathbf{c}$ events) has a significantly different spectral energy profile than the following vowel. Note also the complexity of events near 1000ms corresponding to the glottal stop (labelled /q/ in TIMIT); this segment is due to the word-initial vowel of the word 'outage', and is common in this context. It can be seen in the spectrogram in panel (1) that

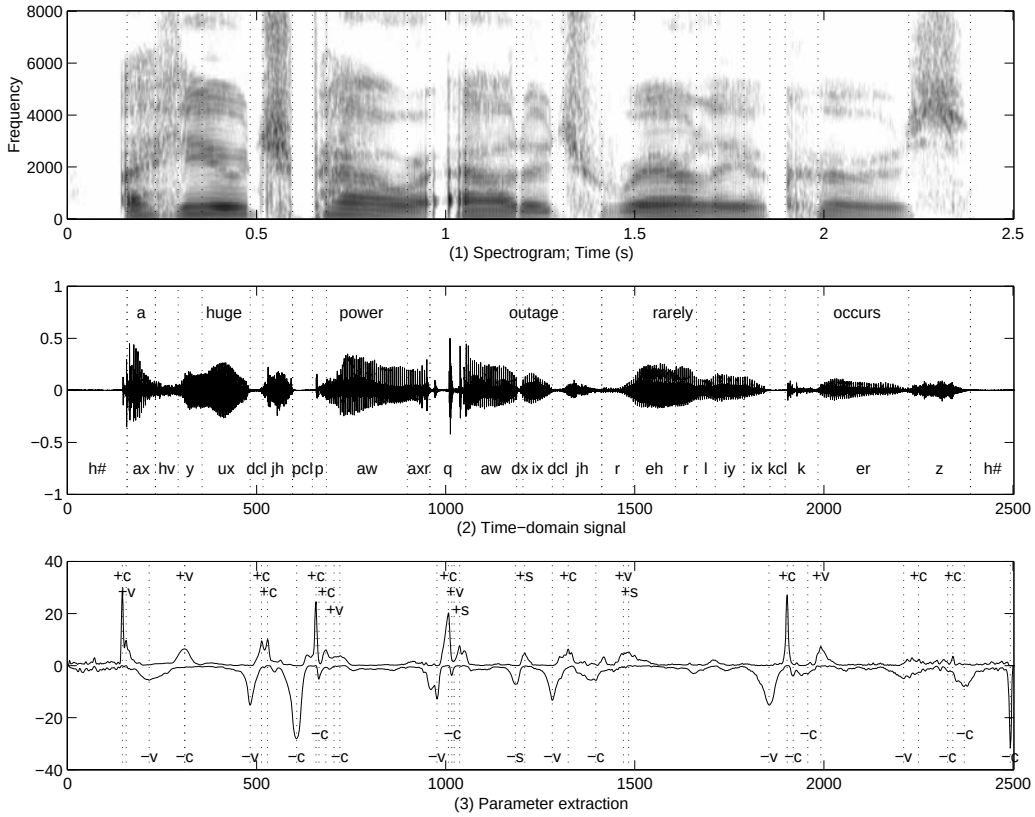

Figure 1.2: Example landmark-labeled utterance: Labels from utterance "A huge power outage rarely occurs," spoken by a female speaker. Segment labels in (2) are from TIMIT database; acoustic event labels in (3) are a set of events corresponding to landmarks in the signal, note relations to parameter signals.

the spectrum of the irregular energy bursts are very similar to the following vowel, unlike the case of the /p/ described above.

1.2 Motivation: Temporal information in the auditory system

This project has been motivated by the study of temporal information used in speech perception and recognition by human and machine. Understanding of spectral contributions to speech perception are well understood. Miller and Nicely [2] showed that low-pass filtering or adding noise in various frequency bands to a signal did not significantly affect perception of gross manner distinctions (such as voicing and nasality), but did severely affect perception of place of articulation. This is evidence for the fact that spectral cues are very important for place, but may be less important for manner features.

More recent research has shown that it is possible to recognize speech from an impoverished signal which conveyed primarily temporal information [3, 4, 5, 6]. The major thrust of this research has been in understanding the types of difficulties that people with hearing deficits have with speech perception, in particular with respect to cochlear implant technology. A variety of processing methods have been employed to produce stimuli used in perceptual tests, but the general result has been that spectrally degraded speech still contains information about manner features. The two studies that will be examined in detail are those from Van Tasell et al. [3] and Shannon et al. [5].

In the earlier study, Van Tasell et al. used speech low-pass filtered with cutoffs at 20Hz, 200Hz, or 2kHz to extract the envelope of the speech signal; this envelope was used to modulate the amplitude of white noise. These degraded signals, which were intended to model the hearing of users of a single-channel cochlear implant, do not have any of the spectral structure of speech but do contain temporal modulations that correspond to the speech signal. A group of 12 subjects listened to stimuli

consisting of /aCa/ syllables (where C is a consonant, e.g. /aba/) containing 19 medial consonants in each of 4 conditions: the 3 degraded versions and a clean version. In each trial, subjects made a decision about which of the 19 consonants in the stimuli set they heard.

Multidimensional scaling analysis applied to the decisions showed that three major distinctions were made by the listeners: 1) a "voicing envelope" parameter, which distinguished voiced consonants /b,d,g,v,dh,z,zh,m,n,r,l,y/ from the unvoiced consonants /p,t,k,f,th,s,sh/; an "amplitude envelope" parameter, which distinguished the sonorant consonants /m,n,r,l,y/ — which have a relatively strong energy profile in this context — from the rest of the consonants; and a "burst envelope" parameter, which distinguished the voiceless stops /p,t,k/ from the rest of the consonants. These parameters correspond to a breakdown of consonants by classes of manner and voicing, although manner distinctions were less strong among the voiced consonants. It was also noted that performance in the task improved significantly from the 20Hz condition to the 200Hz condition, but did not improve significantly from the 200Hz condition to the 2000Hz condition suggesting that the most important temporal information for this task was at fluctuation rates below 200Hz. More detailed analysis with stimuli from multiple speakers, and across both listeners with normal hearing and users of cochlear implants are discussed in [4].

Shannon et al. [5] have also shown that manner classes of speech sounds can be recognized by human listeners based on spectrally degraded speech. In this case, the degraded signal was produced by computing envelope functions over several bands of the speech signal, and then using these envelopes to modulate white noise (filtered by the original bandpass filter used for analysis) in the corresponding bands. Prior to modulation, the envelopes were low-pass filtered at various rates to produce several conditions with different maximum modulation rates. In the multi-band cases (2 bands with cutoff at 800Hz, 3 bands with cutoffs at 800Hz and 1500Hz, or 4 bands with cutoffs at 800Hz, 1500Hz, and 2500Hz), some high-level spectral information

has been preserved; this is intended to model a multi-channel cochlear implant. As before, this processing generates a reconstructed version of the signal where all fine spectral structure in the signal has been removed but information about temporal structure is preserved. Although performance significantly improved from the single channel case to the 2-channel case, there was no significant improvement in recognition of manner and voicing from the 2-channel case to 3 or 4 channels, with the results consistently around 90% information transfer.

This study also noted that decreasing the envelope filter cutoff from 500Hz or 160Hz down to 50Hz caused no significant degradation in recognition performance, though some decrease in performance was seen with an envelope filter at 16Hz; suggesting that temporal accuracy in the range of 20-62.5ms (16-50Hz) and above is critical for speech recognition. Manner and voicing were strongly present in the degraded signal, which suggests that humans can use dynamic temporal information to recognize manner class of speech, and therefore that temporal cues are relevant and robust for detection of manner and voicing.

Considering these results, it should be possible to build a detector for acoustic events that is both largely independent of detailed spectral information and resistant to noise. Further, addition of temporal parameters should also improve performance and increase noise resistance for a system based on spectral information.

As speech perception in humans relies on the information available in the auditory perception system, it is important to consider the types of information available in neurophysiological representations. Two important results have bearing on the work in this thesis. The first relates to the use of bandpass envelope structure as a cue in the auditory system. It has been shown that envelope representations are present in the higher level auditory system, note for instance work by Joris [15, 16, 17] which discusses the use of an envelope model in the process of extracting intra-aural level differences and intra-aural time differences in the task of sound localization. There

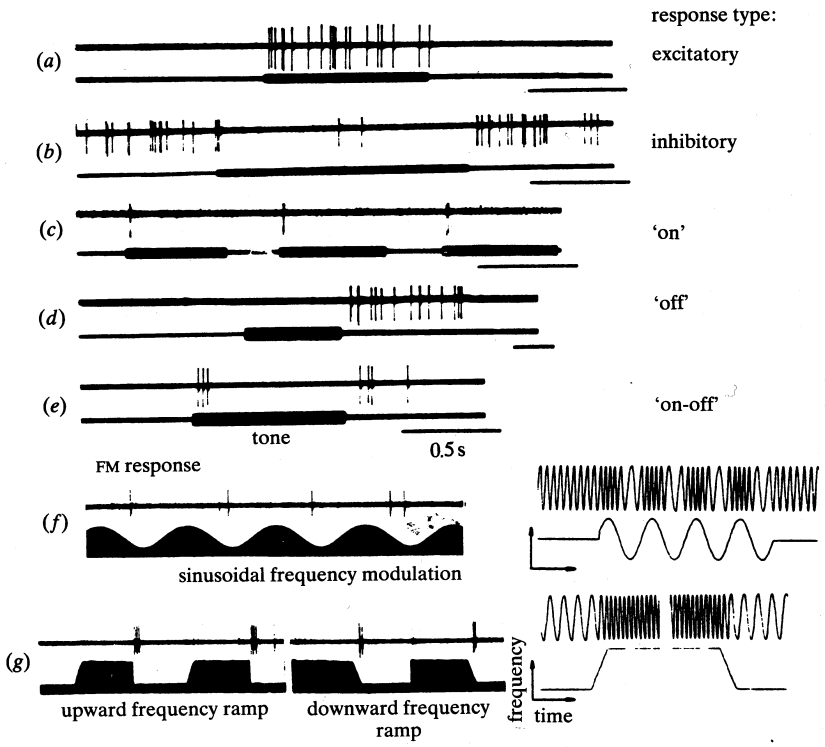

Figure 9. Variety of response types of cells in primary auditory cortex. (*a–e*) Types of response to steady tones. (*f, g*) Cell responding to frequency-modulated, not steady tones. Note response selectivity to direction of frequency sweep: in the downward direction, not to upward sweeps. Black envelope indicates excursions of frequencies illustrated by the waveforms to the right. (From Evans 1968, 1982.)

Figure 1.3: High-level representations in the primary auditory cortex. Note particular responses to onsets (c, e) and offsets (d, e) of tones. Reprinted from Evans [13], originally from [14].

is also evidence of higher-level cortical representations of onset and offset responses in the brain. Note Figure 1.3 from Evans [13, 14] which shows sample responses of a number of cells in the primary auditory cortex.

1.3 Previous work in temporal parameter extraction

In the Speech Communication Laboratory, we have studied another version of the problem of temporal parameter extraction. Our previous work [18] involved extracting a set of temporal parameters for use in a classification task for consonants in intervocalic context. This study involved use of an energy difference operator to extract acoustically abrupt locations in the speech signal, as described in Chapter 2 using a short-time Fourier transform (STFT) for spectral analysis. The events were matched to a standard consonant template via a heuristic algorithm, within a region defined by the transcribed location of the consonant in the signal. The parameters extracted included the strength and time differences among the events. In this study, 87.2% correct manner and 93.8% correct voicing was achieved across a database of 382 sentences containing 1564 intervocalic consonants. Error analysis showed that a more detailed study of the temporal structure of speech and robust algorithms to extract relevant information were needed.

Further work [19] showed that a consonant manner classifier based on event detection from temporal cues can be resistant to spectral degradation (of the same sort used by Shannon et al.) In particular, the classification mechanism was resistant to noise and performed as well as the best human listeners on the same dataset[4], as long as a hand-labeled event extraction was performed. This work again suggested that an event extraction approach could be successful in speech analysis, though focusing on the requirement for accurate underlying detectors.

[4] Experiments with human listeners performed by M. Matthies and L. Davis in the Boston University department of Communication Disorders.

Another approach to use of temporal parameters in speech recognition has been suggested by Sharma and Hermansky [20]. In this method, features are extracted by taking a rather long temporal trajectory in a single frequency band, referred to as a TRAP (TempoRAl Pattern). Both a simple linear correlation classifier, and a multi-layer perceptron (MLP) classifier, applied to the TRAPs showed performance comparable to a baseline classifier based on spectral parameters.

1.4 Goals of the thesis

The major goal of this thesis is reliable extraction of temporal parameters for use in a speech recognition system. In the context of the knowledge-based recognition paradigm, it is proposed that these parameters are important for location of speech landmarks. As such, the system constructed in this thesis uses strictly temporal parameters for location of landmarks in the speech signal. The expected output of the system is a set of potential landmark locations, with some degree of classification as possible from temporal information, such as voicing which is cued by periodicity. Comparison with a set of landmark labels (in the case of this thesis, generated from a transcribed database of speech) will allow judging how well this has been done.

Chapter 2

SIGNAL ANALYSIS

The major algorithm used for event location in this project is an onset/offset detector based on a first difference measure, that was originally derived from the stop burst onset detector by Bitar [21], also used for abrupt event detection by Espy-Wilson [9, 10]. The onset/offset measure is constructed from first differences in each channel output from a spectral analyzer, originally a short-time Fourier transform (STFT), implemented with an FFT. The first difference is computed as a log difference between the sum amplitude of two adjacent non-overlapping windows of the signal in a particular channel, as per the equation

$$D_{i,k}(n) = 20\log\sum_{m=-\infty}^{\infty} x_i(n+m)w_1(m) - 20\log\sum_{m=-\infty}^{\infty} x_i(n+m-k)w_2(m-k) \tag{2.1}$$

where $x_i(n)$ is an individual channel input signal, k is the time difference between the two windows, and the windows $w_{1,2}(n)$ are (usually, rectangular) windows of length $\leq k$. The computed difference is scaled in decibels (dB). This first difference operation is essentially the same as the rate-of-rise (ROR) detector used by Liu [11].

From these per-channel differences, two measures are computed: the positive differences (increasing levels) are summed to produce an 'onset' signal, and the negative differences (decreasing levels) are summed to produce an 'offset' signal. The offset parameter is usually inverted for analysis to make it positive, allowing generalization of all further computations; note that the non-inverted negative version of the parameter is the one shown in all figures. A scaling by $\frac{1}{N}$, where N is the total

number of channels, produces values on a dB scale:

$$on(n) = \frac{1}{N} \sum_{i \in \{i: D_{i,k}(n) > 0\}} D_{i,k}(n) \tag{2.2}$$

$$off(n) = \frac{1}{N} \sum_{i \in \{i: D_{i,k}(n) < 0\}} D_{i,k}(n) \tag{2.3}$$

It was observed that by increasing the window sizes (and correspondingly increasing k, referred to as *difference time*), for the first difference computation, noise in the measurement over the utterance is reduced. This modification of the detector targets slower rates of fluctuation in the temporal envelopes of the signal. This allows a wider range of events to be detected: the full set of abrupt events that we are interested in, rather than just stop bursts. Events are located at peaks (dips) in the onset (offset) parameter. The detector was also modified to produce greater temporal resolution by decreasing the step size from 5ms to 1ms. An unfortunate side effect of lengthening the window sizes was a decrease in the strength of peaks in the onset signal associated with stop bursts, resulting in the development in this thesis of a dynamic method of adjusting difference length based on features of the signal. Several versions of these parameters are shown in Figure 2.1.

In this thesis, the parameter set was increased to include periodicity measurements, as well as improvements to the energy difference operator to sharpen its response in a number of ways, in particular by adapting the difference time according to periodicity information. This enables targeting more of the temporal information in the signal. These two types of parameters—periodicity and energy differences—are then integrated to locate a set of events in the signal, a procedure that is discussed in detail in Chapter 3. The times of these events are derived both from peaks in the onset and offset waveforms, and from additional events generated by the beginning and end of confidently periodic or aperiodic components in the signal.

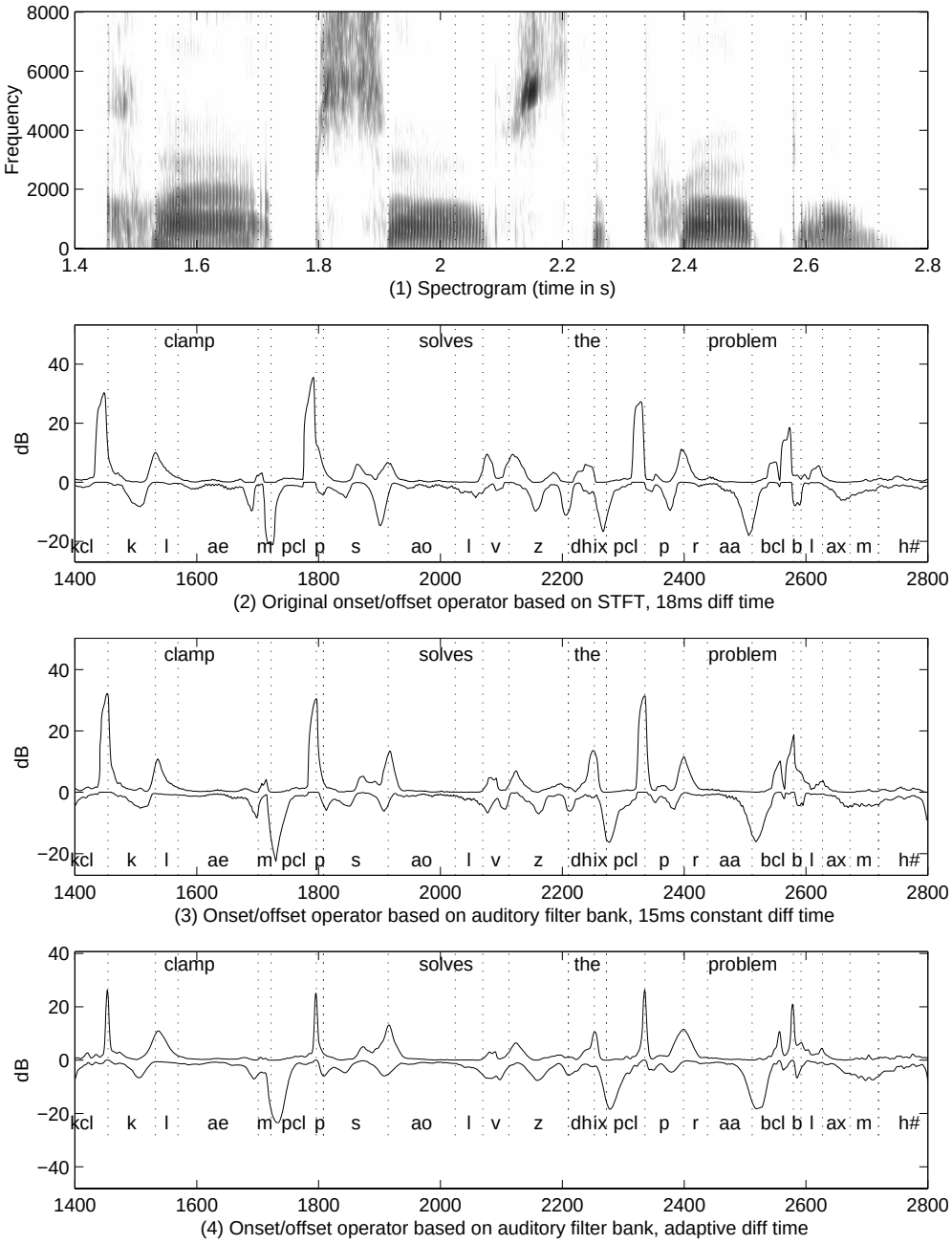

Figure 2.1a: Evolution of the energy difference operator. Top panel is the spectrogram of the utterance ".. clamp solves the problem," spoken by a female speaker. Panel (2) shows the original STFT difference operator; panel (3) is a modified version of the above based on an auditory filter bank front end; and panel (4) is the present dynamic auditory version of the detector.

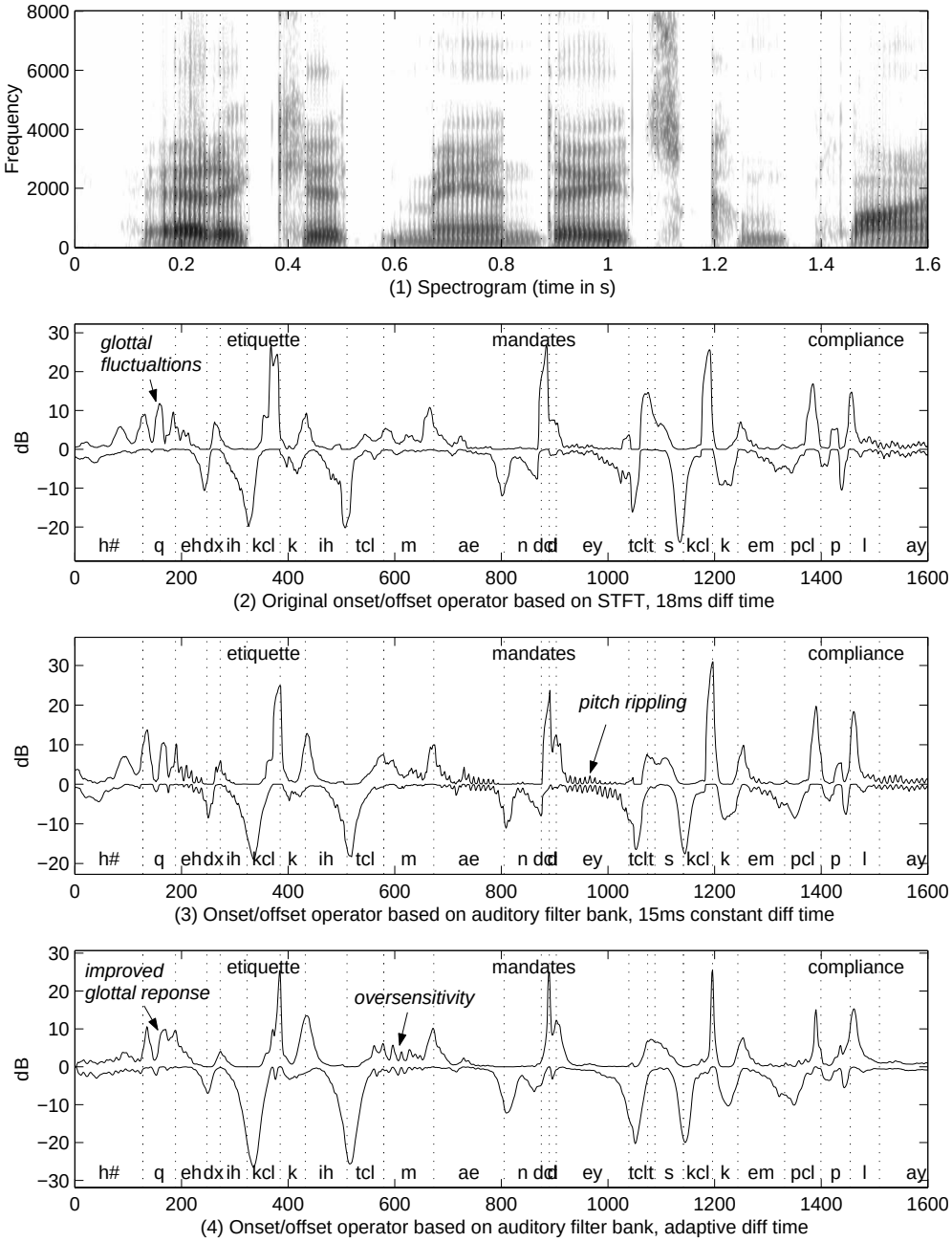

Figure 2.1b: Evolution of the energy difference operator. Difference operators for the utterance "Etiquette mandates compli(ance..)," spoken by a male speaker. Panels as in (a). Note pitch rippling effect due to misalignment of windows with pitch periods, and glottal effects of fluctuation on the onset and offset waveform which is largely absent in the adaptive version in panel (4).

A discussion of the range of temporal information available in the signal is discussed in Section 2.1. Generation of temporal envelopes to be analyzed (filterbank and envelope processing) is discussed in Section 2.2. Computation of pitch and periodicity measurements and corresponding information about aperiodic excitation are developed in Section 2.3. Finally, the current version of the energy difference operation, which includes adaptation to context information from the periodicity subsystem, is described in Section 2.4.

2.1 Defining temporal information

For the purpose of this study, it is helpful to specify exactly what is meant by the "temporal information" in the signal. In this study, we define temporal information as the structure of envelopes of bandpass components of the speech signal. This is especially important in terms of characterizing the results of the previous work, which suggested that not all temporal information we were interested in targeting was being captured. A productive system of categorization for temporal information has been suggested by Rosen [22], who proposes three categories of temporal information in speech: (1) "envelope information" (with fluctuations at rates from 2 to 50Hz) which contains amplitude and duration cues to manner of articulation and voicing, as well as information about vowel identity (e.g. vowel length) and prosodic cues; (2) "periodicity information" (fluctuations at rates from approximately 50 to 500Hz) provides cues to voicing which can aid in manner identification, as well as marking stress locations by changes in pitch; and (3) "fine structure" (fluctuations at higher rates) which provides information about spectral shape which is most useful for identifying place of articulation for consonants and vowel quality, though there are some cues for manner such as the high-frequency content of many obstruents. These categories imply that use of temporal information for a recognizer (particularly when combined with spectral cues) should concentrate on two major types

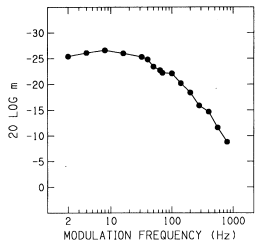

FIG. 4.1 A temporal modulation transfer function (TMTF). A broadband white noise was sinusoidally amplitude modulated, and the threshold amount of modulation required for detection is plotted as a function of modulation rate. The amount of modulation is specified as 20 log(m), where m is the modulation index (see Chapter 3, section 2E). The higher the sensitivity to modulation, the more negative is this quantity. Adapted from Bacon and Viemeister (1985b).

Figure 2.2: A temporal modulation transfer function [23], adapted from [24].

of information: low-frequency envelope, and periodicity associated with pitch. The low-frequency amplitude envelope (or more specifically, the sharpest changes in this envelope) are captured in this work by an energy difference measure.

Note that according to Viemeister [25, 24], normal-hearing subjects cannot detect amplitude fluctuations above about 1000Hz; and that response to modulation degrades rapidly above 100Hz (see figure 2.2). This suggests that at most human listeners can only derive first-formant information from the temporal fine structure, and possibly that no information regarding fluctuations above the rate of pitch modulations are perceptually significant. Further, there is evidence [26] which suggests that particular ranges of low-frequency envelope information are relevant for particular types of distinctions, in particular with respect to energy fluctuation rates below 30Hz.

2.2 Filterbank and envelope processing

A number of options exist for spectral analysis and envelope processing. The version of the energy difference method used in earlier work [18, 19] used a STFT which has characteristics of linear scaling in frequency, and constant bandwidths among channels. Smoothing in this system was provided by the windows (6ms Hamming) used

prior to the Fourier transform. Due to the pre-STFT windowing (6ms Hamming), the windows in Equation 2.1 (applied over the per-ms STFT samples) were chosen to be slightly shorter (15ms) than the difference length of 18ms. This processing method was determined to be less than optimal, noting two specific problems: frequency scaling and the level of smoothing. In terms of frequency scaling, the human auditory system is much more sensitive in the range of speech to low frequencies (near the first formant and fundamental) than higher frequencies due to the approximately logarithmic scaling of frequency along the cochlea (particularly for high frequencies), whereas the STFT difference measure was overly sensitive to high frequency information. Additionally, in the time domain, the windowing method in the earlier work was overly smoothed, as each window in the difference measure was computed from approximately 20ms of the input signal between the hamming window and rectangular averaging over the window.

The particular envelopes which are used in this work are generated from the outputs of an auditory γ-tone filterbank with characteristic frequencies (CFs) based on physiological data as per Carney [27, 28]. Note that the CFs, which are listed in Table 2.1 for the 60-channel version of the model used in this thesis, are roughly linearly spaced at low frequencies and logarithmically spaced at higher frequencies; and also that the bandwidths are approximately constant-Q. An auditory filter bank was chosen for spectral analysis in order to provide an accurate weighting of frequency components, most importantly in terms of the strength of events corresponding to voicing excitation of speech relative to their unvoiced counterparts. A sample set of envelope outputs for an utterance are shown in Figure 2.3a; note detailed views of a voiced region and an unvoiced region in Figure 2.3b.

As a particular example of the weighting issue, note the example of a vowel as compared with a fricative. The original STFT linear filterbank was overly sensitive to unvoiced sounds such as the fricative /sh/ or the africate /j/, which can be seen at an offset of 550ms in Figure 2.3a. These segments usually exhibit a strong energy

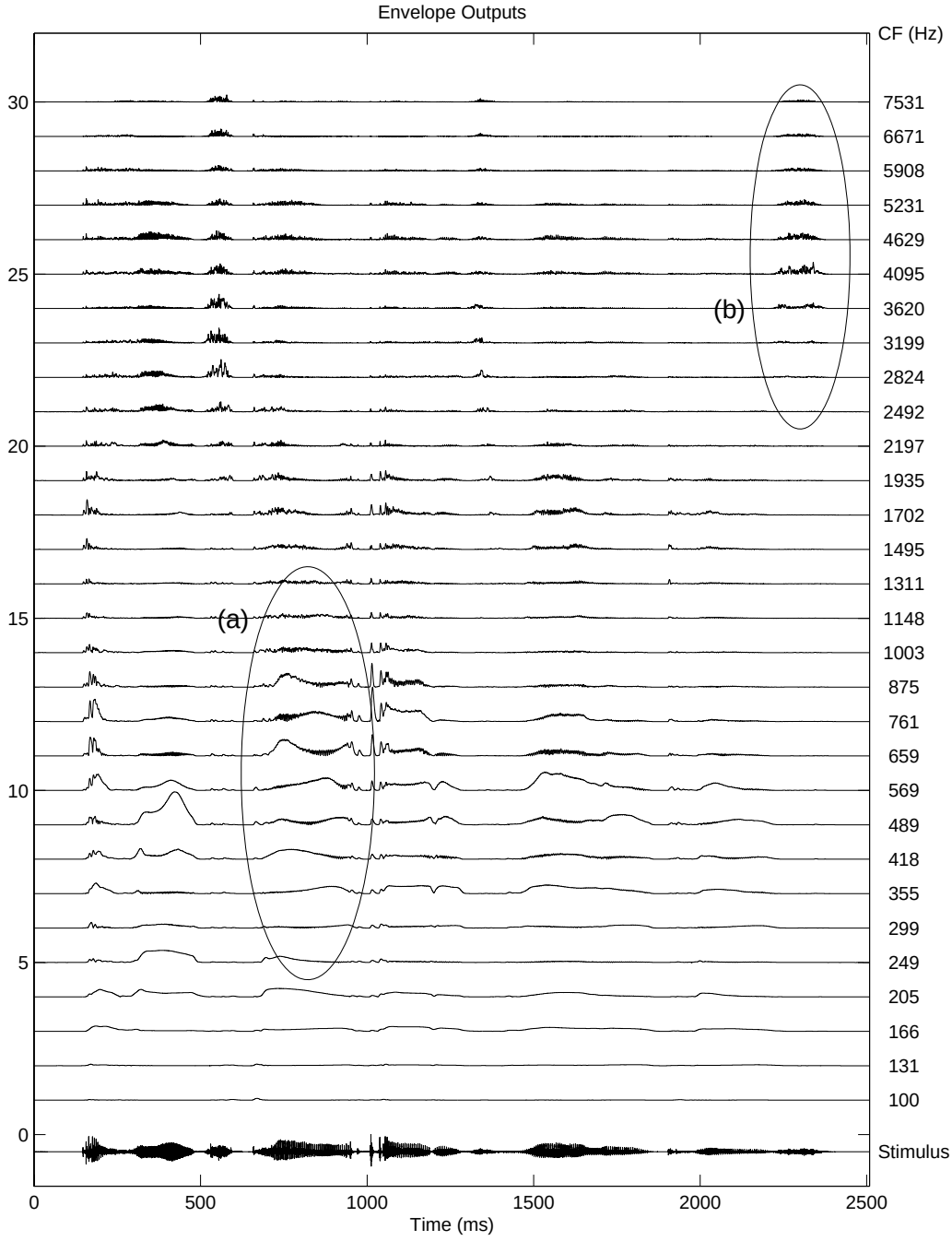

Figure 2.3a: Output channels from envelope processing, for the utterance "A huge power outage rarely occurs," spoken by a female speaker (note that only every second channel is shown). Close-ups of the marked regions are shown in Figure 2.3b.

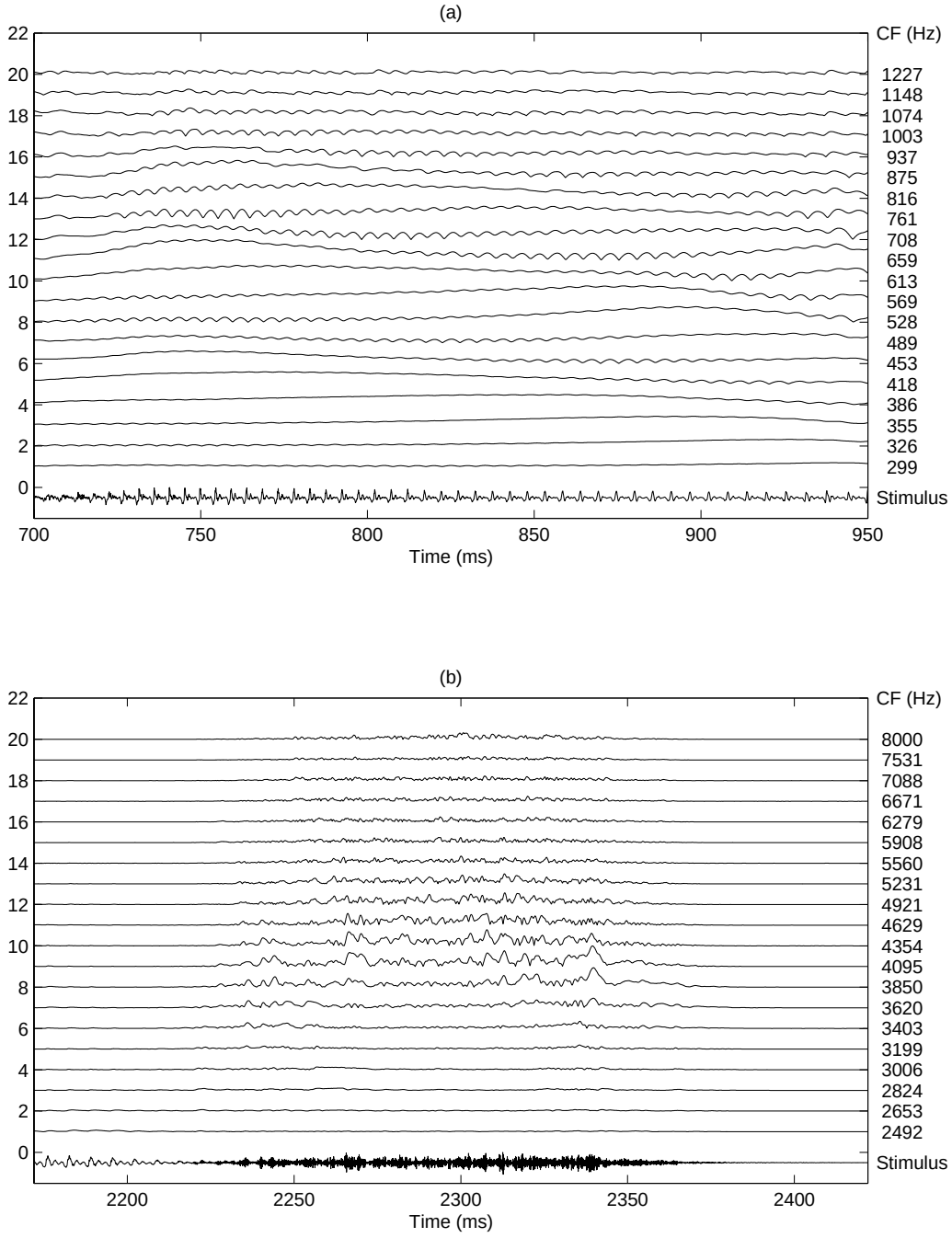

Figure 2.3b: Output channels from envelope processing for two regions in the utterance. Part (a) shows the explicit voicing in low frequency channels during a vowel; part (b) shows unvoiced excitation in high frequencies which is clearly aperiodic.

Table 2.1: CFs of auditory filter bank channels

#	CF	#	CF	#	CF	#	CF	#	CF	#	CF
1	100	11	299	21	659	31	1311	41	2492	51	4629
2	115	12	326	22	708	32	1400	42	2653	52	4921
3	131	13	355	23	761	33	1495	43	2824	53	5231
4	148	14	386	24	816	34	1595	44	3006	54	5560
5	166	15	418	25	875	35	1702	45	3199	55	5908
6	185	16	453	26	937	36	1815	46	3403	56	6279
7	205	17	489	27	1003	37	1935	47	3620	57	6671
8	226	18	528	28	1074	38	2062	48	3850	58	7088
9	249	19	569	29	1148	39	2197	49	4095	59	7531
10	273	20	613	30	1227	40	2340	50	4354	60	8000

Characteristic frequencies (CF) are in Hz.

profile over more than half of the spectrum under consideration, from approximately 2500Hz up to the Nyquist rate of 8kHz (as the sampling rate in the TIMIT corpus is 16kHz). Note that the onset of such a segment is certainly not more and likely somewhat less perceptually significant than a vowel which may be strong in less than half as much of the linear spectrum, e.g. from 100Hz up to 2500Hz (a range which will usually contain the excitation frequency and the first three formants). The auditory filter bank, on the other hand, contains twice as many channels from 100-2400Hz (40 channels) as from 2400-8000Hz (20 channels). It is also worth noting that most vowels will have sharp dips in the spectrum between formant frequencies, and will exhibit most energy near harmonic frequencies (and essentially no energy between them and below the fundamental pitch); as such, a number of channels among these 40 will not be strongly excited.

In order to avoid excessive smoothing in the time domain, the simple linear smoothing (spectrum averaging implicit in the windowing operation for the difference operator) was replaced by an envelope operator based on the Hilbert information. The envelopes $e_i(t)$ of the individual channels are obtained by the function

$$e_i(t) = |x_i(t) + j \cdot H\{x_i(t)\}| \tag{2.4}$$

where $x_i(t)$ is the input signal, and $H\{x_i(t)\}$ is the Hilbert transform [29] of the input signal. Given a real bandpass signal as input, the Hilbert transform produces a version of its input signal that is precisely 90° out of phase, such that the amplitude of the complex sum of these two signals is an estimate of the low-frequency amplitude modulation applied to the signal (it is also possible to obtain the instantaneous frequency, i.e. the frequency modulation applied to the narrowband signal, by looking at the phase component of the complex sum).

This transform is an improvement over a simple smoothing or filtering because abrupt changes are preserved, at the maximum rate that can be captured by the particular channel given its CF. It also has the advantage of preserving periodicity of amplitude modulation, especially when that rate is significantly slower than the CF—for example the fundamental frequency of voiced speech as compared with much of the spectrum in the range of the formant frequencies (especially above the first formant, which can be close to the fundamental). It is important to be able to detect periodicity in these higher frequency channels.

2.3 Periodicity analysis

An estimate of the pitch, the fundamental frequency of vibration corresponding to voiced excitation, of the speech signal is computed by producing estimates of the fundamental period in every individual channel and combining by way of a modified histogram. By producing estimates independently in all channels, the degree of periodic excitation can be determined by counting the number of channels that

agree with the determined pitch, weighted by confidence. The degree of aperiodic excitation is computed by counting the number of non-silent channels (above a silence threshold, as discussed below) that either do not appear to be periodic, or do not agree with the primary pitch estimate.

The measure used to generate raw estimates of pitch period, and a corresponding confidence of periodicity, is the short-time Average Magnitude Difference Function (AMDF), as defined in Rabiner & Schafer [30]:

$$\gamma_n(k) = \sum_{m=-\infty}^{\infty} |x(n+m)w(m) - x(n+m-k)w(m-k)| \tag{2.5}$$

where $x(n)$ is the input signal, and $w(m)$ in this case is a 20ms rectangular window. This function looks roughly like an inverted autocorrelation function, as shown in Figure 2.4. The pitch period estimate is produced by choosing the sharpest dip in the function, which is the point where windowed segments are most similar.

In each channel, periodicity estimates are made approximately once every pitch period using the AMDF measure. At each estimate location, a window of roughly the length of the current pitch period is first tested for silence by comparing the maximum value in the region with a constant threshold (approximately 66dB below the maximum signal level[1]). If this region is below the silence threshold, no AMDF measure is taken as the channel is considered silent at that point. Otherwise, an AMDF measure $\gamma_n(k)$ is taken and the pitch estimate is extracted by finding the maximum dip in the AMDF using a convex hull $h(k)$ (from left to right, i.e. low k to high k). The confidence of a pitch estimate is taken as the depth of the dip relative to the hull ($\frac{h(k)-\gamma_n(k)}{h(k)}$), as shown by the dotted lines in Figure 2.4. The chosen pitch period is the most confident dip according to this measure.

Periodicity estimates are combined across channels using a modified histogram

[1] Sample values from -32768–32767 are scaled by a factor of 10^{-6}, giving a dynamic range of -29.7dB; the threshold used at this stage in the trained system was -90dB relative to the reference value of 1.0, giving a silence threshold of roughly 60.3dB from the theoretical maximum.

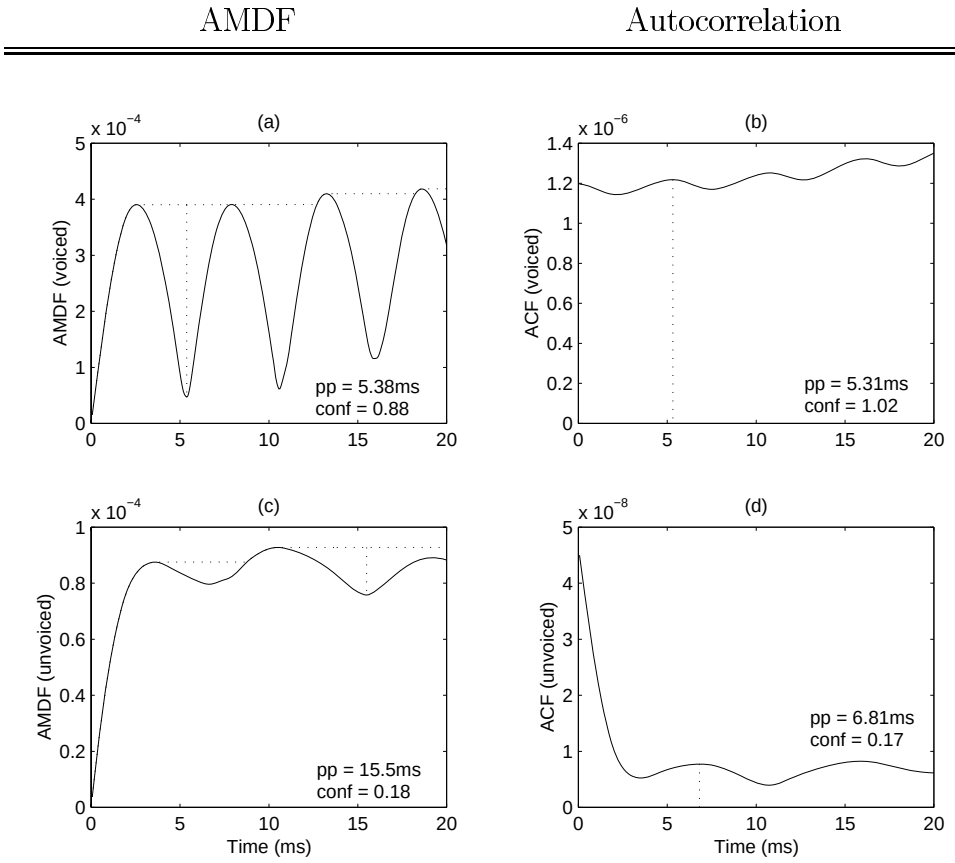

Figure 2.4: Average Magnitude Difference Function (vs. autocorrelation). The Average Magnitude Difference Function (AMDF) and Autocorrelation (ACF) for the same segments of voiced and unvoiced speech in an individual channel. Note that the functions are shaped similarly but with an inverted sense of the vertical axis. AMDF measurements are augmented with a dotted line showing the convex hull where it is different from the function, and a vertical line at the chosen pitch period. Note also the confidence metrics and pitch estimates below. In the AMDF, the pitch estimate is the most confident dip, measured by *(depth of dip) / (hull height)*. In the autocorrelation function, the pitch estimate is the first peak, with the voicing confidence measure *(peak height) / ACF(1)*, i.e. peak height scaled by lag-1 autocorrelation.

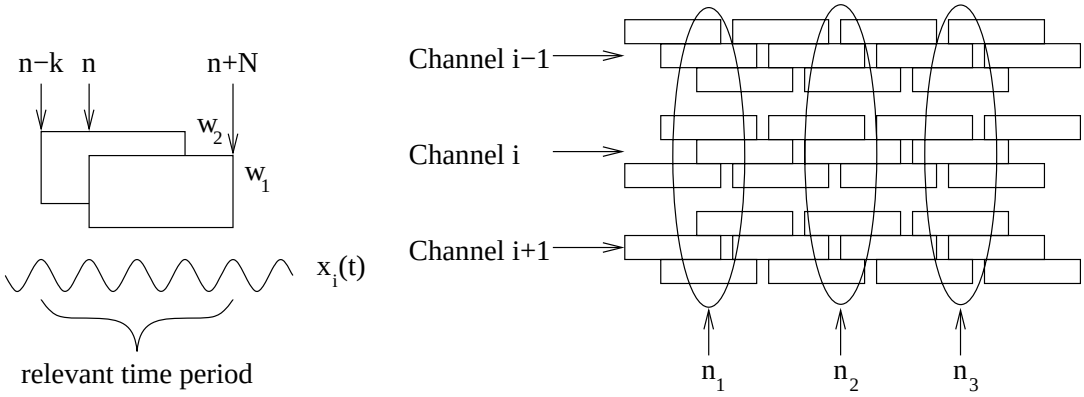

Figure 2.5: Selection of pitch estimates. In (a), note that the relevant time period for a pitch estimate starts at the beginning of w_2 (which is at time $n-k$) and ends at the end of w_1 (which is at time $n+N$, where N is the length of the window). In (b), each pair of windows denotes a pitch estimate (perhaps all in a single channel). Note that by the definition above multiple relevant pitch estimates are available at each point labeled with an $\uparrow$ in (b).

across all relevant estimates. This set is defined by considering each estimate as relevant for a period from one pitch period back from the estimate time (the lag in the AMDF used to produce the estimate), to a distance of the length of the window (20ms) forward, illustrated by Figure 2.5a). As such the set of relevant estimates at time step n contains all estimates such that the time under analysis is present in this region of relevance for the estimate, as shown in Figure 2.5b. Each contribution to the histogram is scaled by the confidence value of the AMDF measurement and only values with a confidence greater than 0.3 are considered. The histogram over sample increment values is smoothed with a 15-sample wide (0.94ms) Hamming window, as shown for both voiced and unvoiced speech in Figure 2.6. Note that often most estimates in the voiced case were within 2 samples of the chosen peak, along with

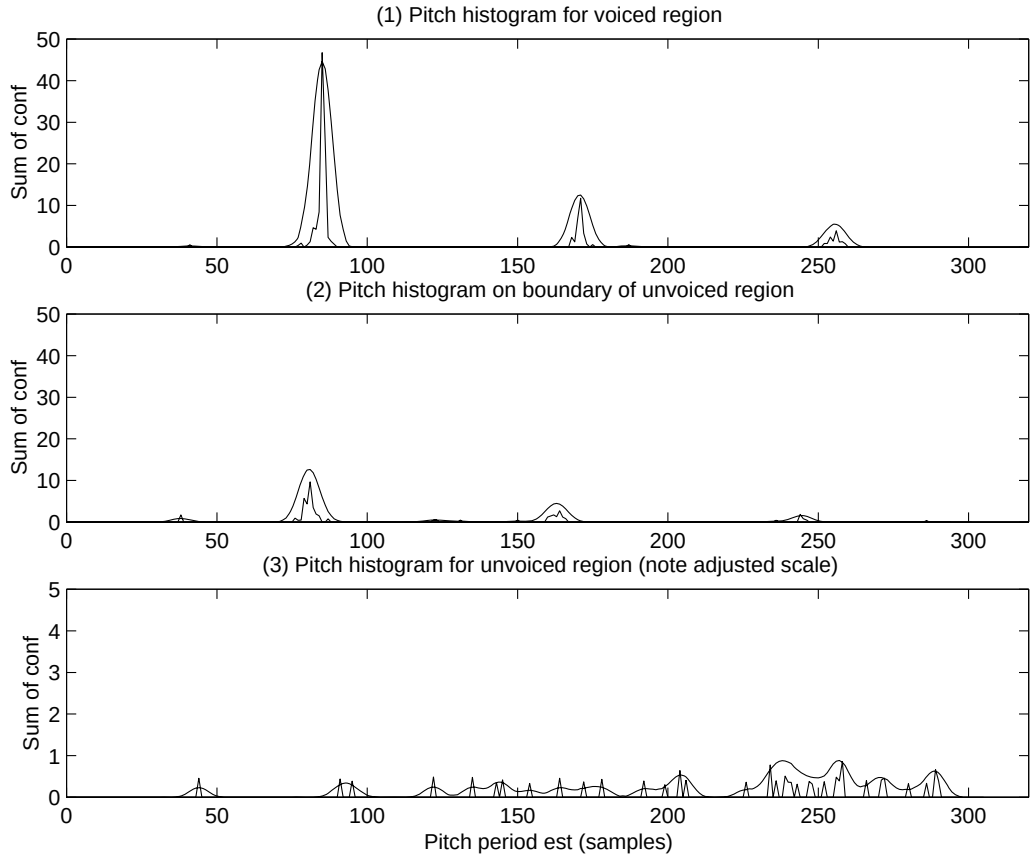

Figure 2.6: Pitch estimate histogram processing. Pitch period estimates are combined using a modified histogram. Plotted here are the raw histogram (sum of confidence measures at each sample delay), with a superimposed Hamming-smoothed version that is used for the decision. Note that confidence measures are computed separately after this decision has been made.

minor peaks at multiples of the pitch period corresponding to channels where an AMDF dip corresponding to a subharmonic was chosen.

A summary periodicity confidence measure P_{conf} is generated by summing the confidence measures of channels near the chosen pitch (or at a multiple of 2). Correspondingly, the summary aperiodicity measure AP_{conf} is a simple count of channels that are not silent, but either have a very low confidence of periodicity (< 0.3) or are not in agreement with the selected pitch period. These are shown in panel (2) of Figures 2.7 and 2.8. The cleaner region-oriented measures in panel (3) are produced by median smoothing and applying thresholds, along with a pruning procedure that requires a maximum peak value for each confident region (discussed in more detail in Chapter 3).

2.4 Energy difference operator

The spectral energy difference operator computes a rate of spectral change. The version used in this thesis has been improved by modifying it such that rather than a constant difference time, the difference time (and corresponding window lengths) are adapted dynamically according to information extracted by the periodicity and silence detection systems. The energy difference detector is adapted in each channel independently, with difference length targets based on the existence of silence or periodic/aperiodic excitation, and according to the pitch estimate in periodic regions. Rules are as follows:

- Difference time is shortened (5ms) for silence, to sharpen response to onsets from silence (for example stop bursts).

- Difference time is lengthened (30ms) in aperiodic regions, to maximally smooth the first difference output in fricated regions.

- Difference time is tuned to exactly twice the pitch period in periodic regions,

to prevent detection of spurious energy fluctuation due to misalignment with the pitch period (what is referred to as pitch ripple in Figure 2.1).

There is also a slew rate control of 0.5ms per millisecond (the difference operator is sampled every ms) to prevent discontinuities.

This set of parameters in combination over a speech signal visibly provide useful information about the content of the signal, as can be seen in Figures 2.7 and 2.8. Note that the periodicity and aperiodicity confidence measures provide a decomposition of the signal into periodic (roughly, voiced) and aperiodic pieces. Also note that the onset and offset measures have peaks at many of the important events in the signal. The next stage of processing is automated extraction of events, a procedure that is discussed in the following chapter.

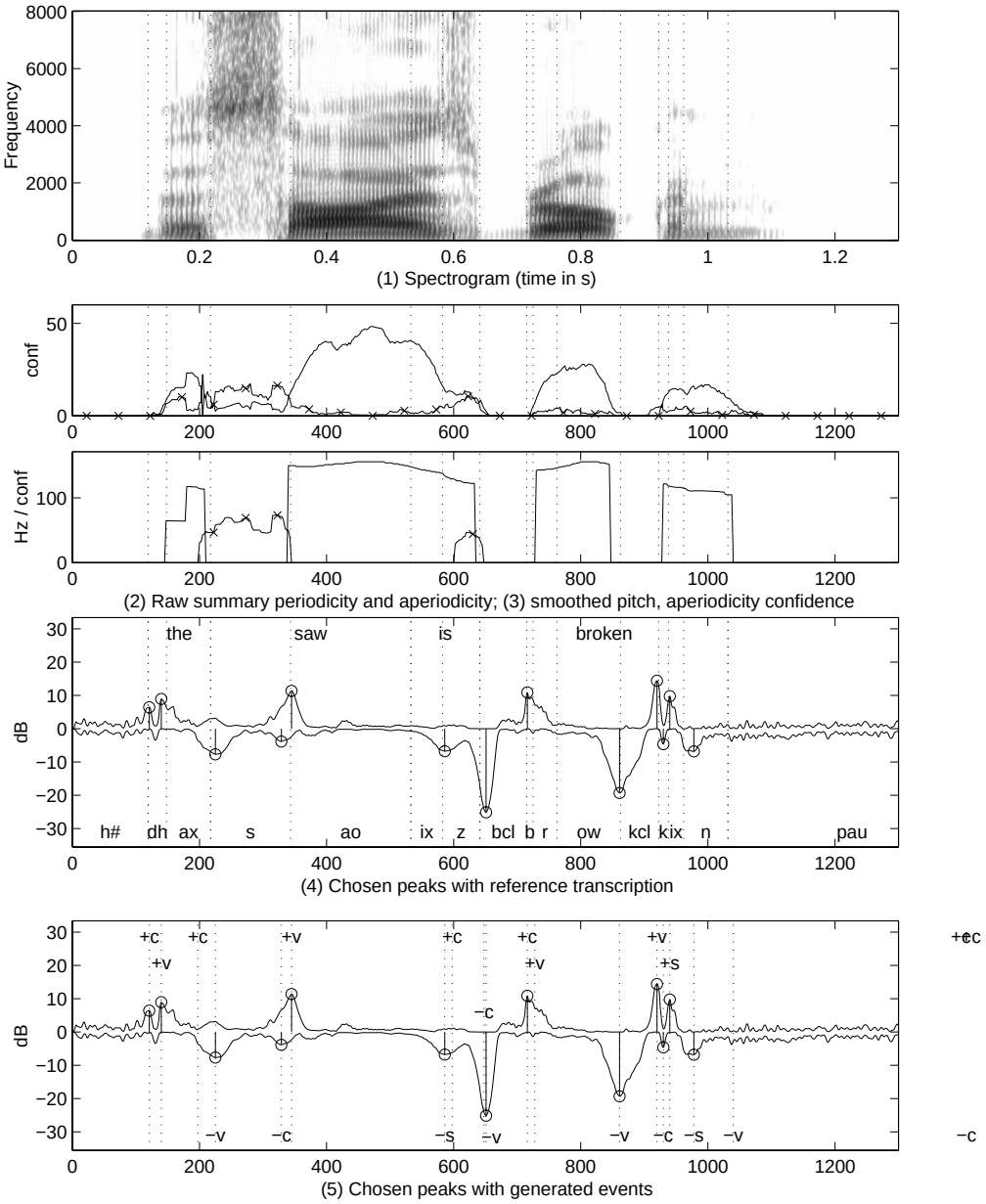

Figure 2.7a: Parameter extraction results (male speaker). Panel (1) is the spectrogram of the utterance, "The saw is broken, ..." Vertical lines mark labeled phoneme boundaries. Panel (2) shows raw summary periodicity and aperiodicity confidence scores (marked with 'x'). Panel (3) shows smoothed pitch in confidently periodic regions, aperiodicity confidence ('x') in confidently aperiodic regions. (cont.)

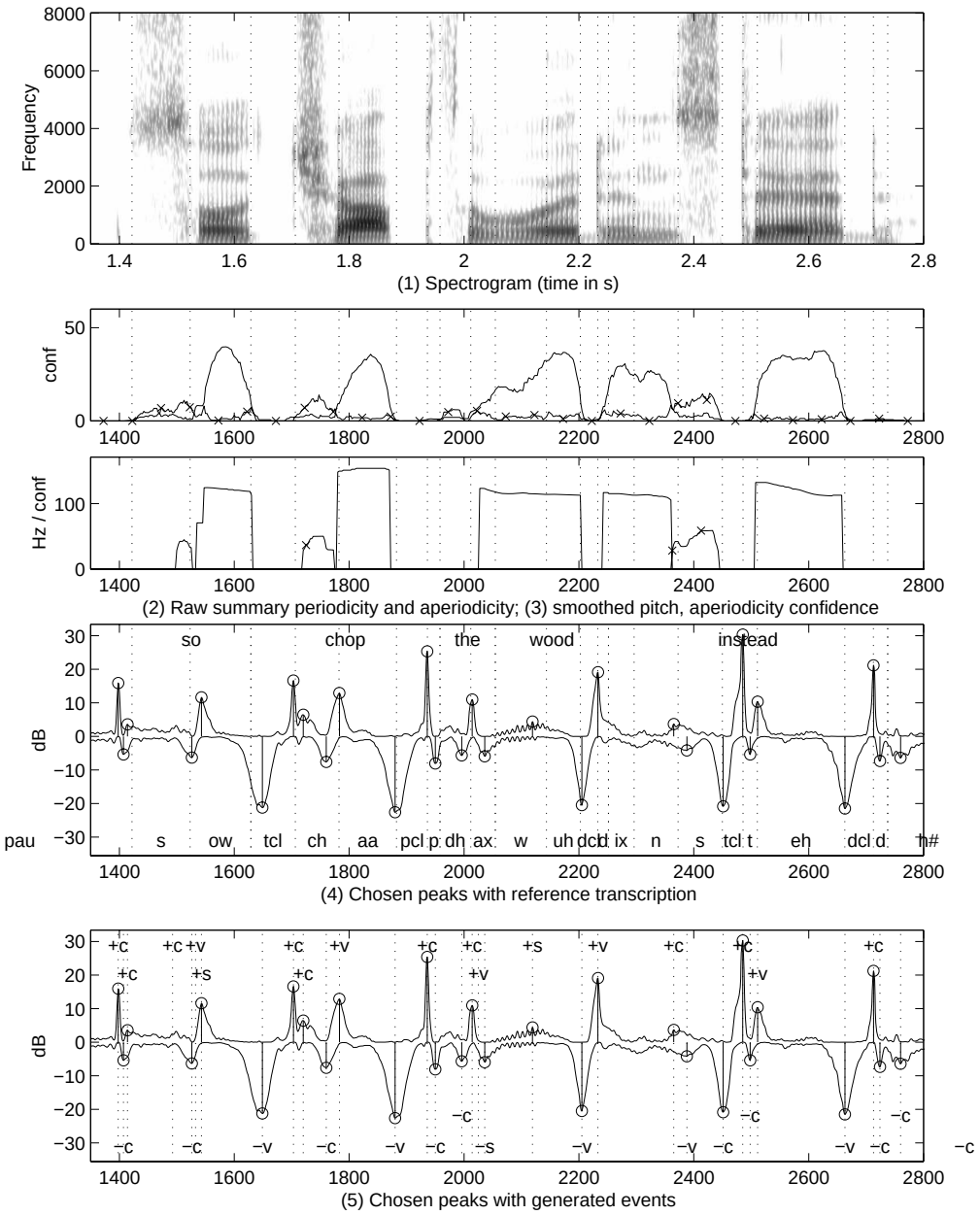

Figure 2.7b: Parameter extraction results (male speaker, cont.) Continuation of utterance, " ... so chop the wood instead." Panels (1)-(3) as in 2.7a. Panel (4) contains the onset and offset signals with chosen peaks labeled with stems. Labels are superimposed on the detector output signals, both the words (top) and phonemes (bottom). Panel (5) contains detected events.

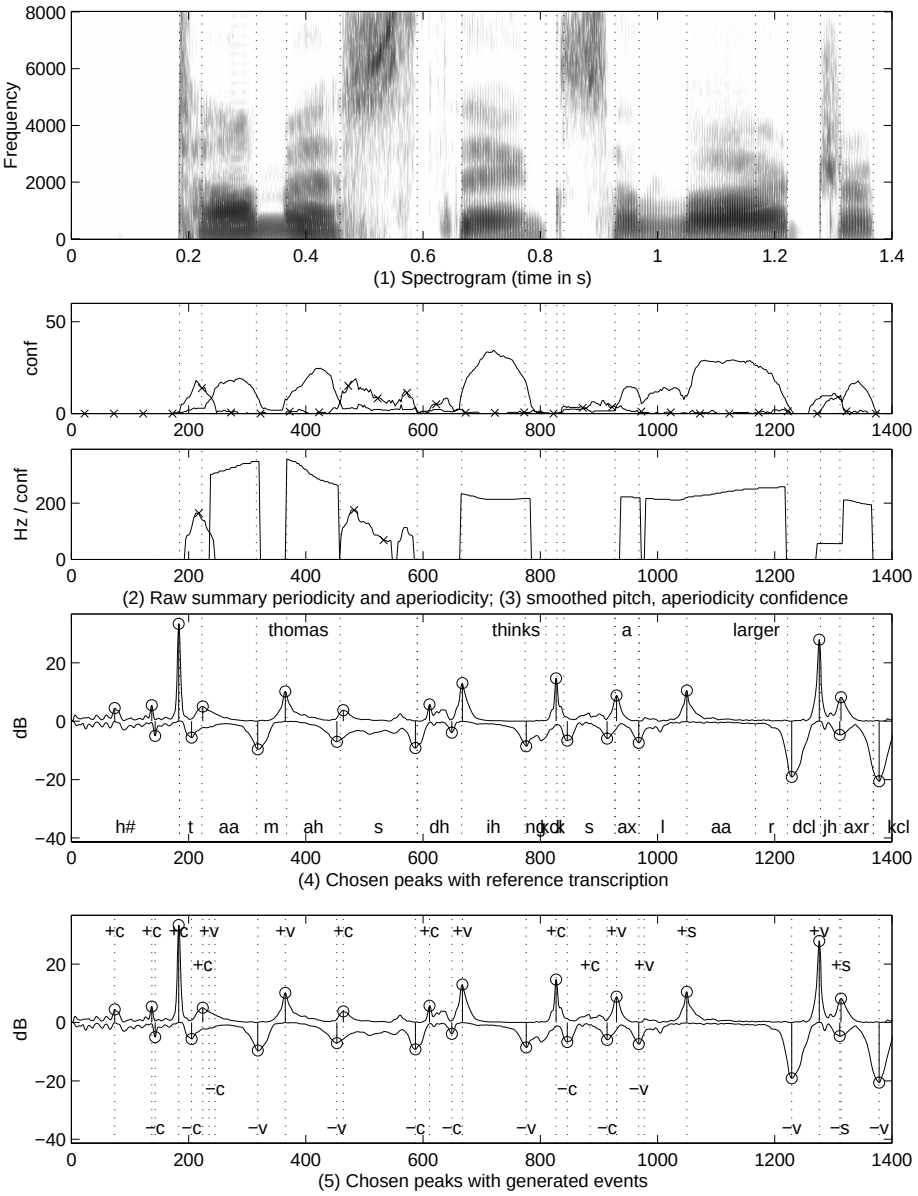

Figure 2.8a: Parameter extraction results (female speaker). Panel (1) is the spectrogram of the utterance, "Thomas thinks a larger clamp ...," spoken by a female speaker. Panels as in Figure 2.7.

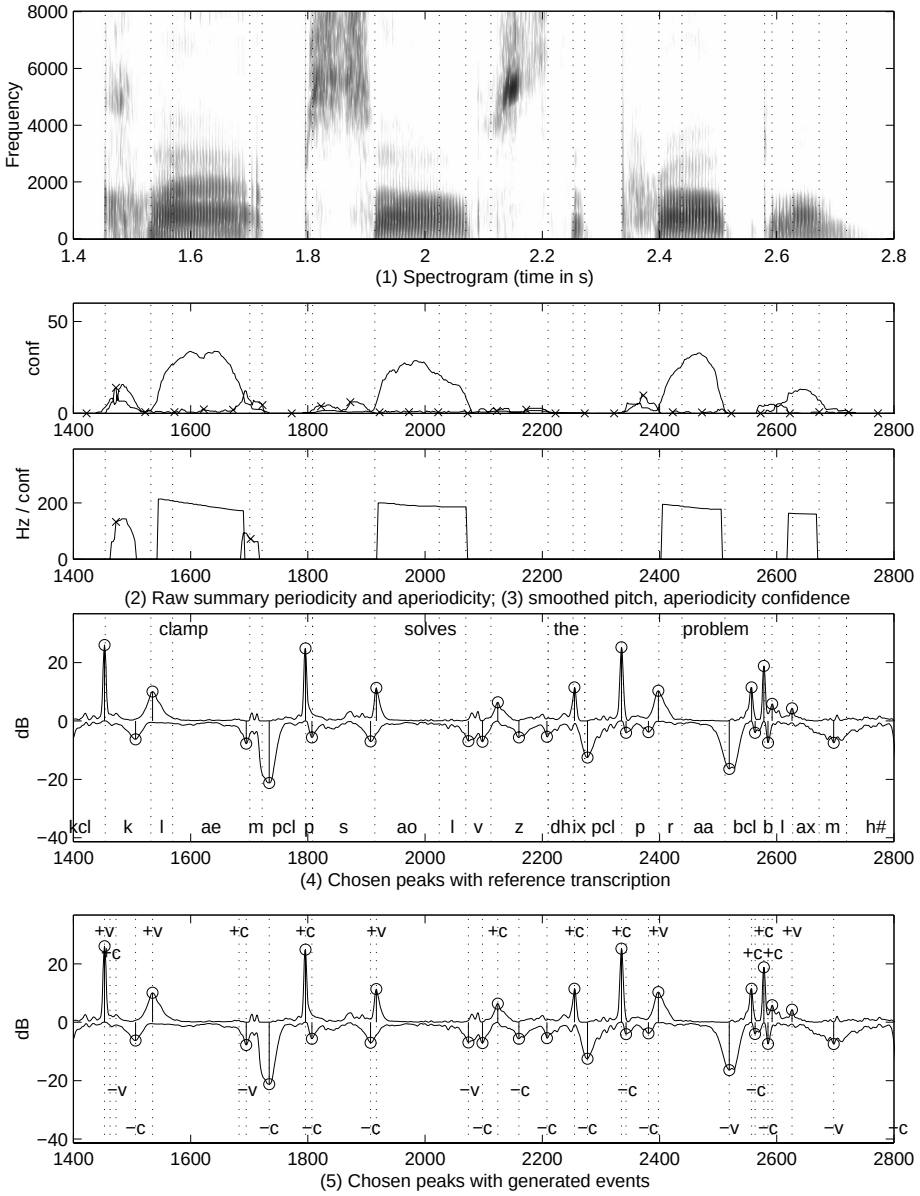

Figure 2.8b: Parameter extraction results (female speaker, cont.) Continuation of the utterance, "... solves the problem." Panels as in Figure 2.7.

Chapter 3

EVENT DETECTION

The goal of event detection is to generate a set of proto-landmarks—referred to as *events*—that will direct further analysis of the speech signal. To ensure the success of further levels of processing (outside the scope of this thesis), this set should be reasonably complete with respect to the perceptually sharpest events, for example events corresponding to stop consonant bursts, strident fricatives, and stressed vowels. Note that insertions will be discarded by further analysis. On the other hand, it is likely that some weaker events are going to be captured less often: semivowels (particularly the glides /w/ and /y/), for which the primary cues consist of formant movement; weak fricatives which have become sonorant, such as a common pronunciation of the /v/ in "everyday" [31]; and other cases of events that do not involve a significant degree of energy fluctuation. In cases of heavily coarticulated segments, it is expected that the output of the system will reflect the type of events that actually occurred rather than the canonical events expected from segment-based labels (e.g. sonorant events rather than onset and offset of frication for the /v/ above). Some unusual cases that are hoped to be detectable are boundaries between segments with the same manner, particularly adjacent fricatives or adjacent vowels, though they are most often unlikely to have strong events of the types targeted by the signal analysis.

3.1 Defining the problem

For the purpose of detection, a set of event types based on acoustic parameters have been defined, and are listed in Table 3.1. The categories correspond to the polarity (onset or offset of energy) of the event, and their correlation with periodic and/or aperiodic excitation. Events are labelled based on their occurrence either at a boundary where periodic content begins or ends ($\pm$**v**, correlated with voicing onset or offset), surrounded by periodic excitation ($\pm$**s**, correlated with sonorant consonant boundaries), or occurrence at a boundary of aperiodic excitation or at least occurring outside of periodic excitation ($\pm$**c**, correlated with obstruent consonants). The output of the event detector consists of this set of event labels. The example utterance in Figure 1.2 shows a sample use of these labels as applied to a real speech signal.

3.2 Landmark extraction algorithm

The two types of low-level information discussed in Chapter 2, periodicity and onset/offset, are combined to locate events. Summary periodicity and aperiodicity confidence measures are analyzed (after median smoothing) to locate potential confident regions and their boundaries. In the first pass, these boundaries are associated with peaks in the onset or offset parameters (depending on the type of boundary) to locate a subset of events that are indicated from both information sources. A second pass generates event labels for those events that are only evident from a single source.

The set of confidently periodic/aperiodic regions is determined by applying a minimum threshold for the maximum P_{conf} or AP_{conf} for a region. Following this, a lower threshold is used to find the boundaries of each region. These thresholds have been trained, and there is a separate pair of thresholds for the two types of regions. There is also some pruning of regions to clean up the output of the periodicity

Table 3.1: Event types

Label	Name	Description *(examples)*
+v	voicing onset	onset corresponding to beginning of periodicity *(beginning of a vowel or sonorant consonant)*
−v	voicing offset	offset corresponding to end of periodicity *(end of a vowel or sonorant consonant)*
+s	sonorant onset	onset within periodic region *(onset at release of nasal or semivowel)*
−s	sonorant offset	offset within periodic region *(offset at closure for nasal or semivowel)*
+c	obstruent onset	onset corresponding to beginning of aperiodicity *(stop consonant burst, affricate or fricative onset)*
−c	obstruent offset	offset corresponding to end of aperiodicity *(stop, affricate or fricative offset)*

detector. Periodic regions are discarded if the pitch within the region doesn't agree with the median pitch of the utterance (off by more than a factor of 2). Aperiodic regions are discarded unless at least one end of the region is associated with an onset/offset event, i.e. the beginning of the region near an onset event or the end of the region near an offset event. They are also discarded if the surrounding region is completely periodic (as there is often a spurious aperiodic response at abrupt boundaries within sonorant regions, e.g. nasal closures and releases), or if the region is shorter than 10ms.

The onset and offset parameters are converted into a sequence of potential events by use of a convex hull-based peak-picking algorithm. There are thresholds for minimum peak height, and a required minimum dip between two adjacent peaks. These thresholds were also trained; there were two independent sets for peaks in the

onset and offset parameters, for a total of 4 parameters.

Onset and offset peaks are associated with boundaries of periodic/aperiodic regions in order to classify event types. Onset/offset peaks located near the beginning/end of a periodic region are labeled as $\pm$**v**. Correspondingly, onset/offset peaks located near the beginning/end of an aperiodic region are labeled as $\pm$**c**. The criterion for this locality is determined by another set of trained thresholds. Remaining boundaries of confidently periodic/aperiodic regions are labeled as landmarks of the same types but the times are less accurate, as the periodicity results are compiled only every 2.5ms (and the underlying AMDF measurements are made even less often, roughly once per pitch period), whereas onset/offset parameters are computed with a 1ms frame rate and are a type of measure that is inherently more accurate in time. Remaining onset/offset peaks are labeled as $\pm$**s** if they are within a periodic region, or $\pm$**c** if they are outside of any periodic region. The full set of trained parameters used in this process are listed in detail in Table 4.1.

3.3 Positing landmarks from phoneme labels

For the purpose of comparing with the reference transcription, a set of expected landmarks was generated from the phoneme-labeled transcriptions available in the TIMIT database. These were generated using a simple rule-based algorithm based on manner class of adjacent segments at each boundary, and are expected to have some inherent error due to the decreased level of information available from the phonetically-labelled level of representation available in the TIMIT corpus. Some of this underspecification is accounted for by inserting events which are labeled as 'non-required' because they are possible, and should be caught by the matching algorithm, but not necessarily strongly expected. The rules are as follows:

- At a boundary between a sonorant (vowel or sonorant consonant) segment and a nonsonorant (including silence such as a stop closure; or a stop, frica-

tive or africate): the system posits a $-\mathbf{v}$ event if the sonorant is before the nonsonorant, or a $+\mathbf{v}$ event if the sonorant segment follows the nonsonorant segment.

- At a boundary between a sonorant consonant and a vowel, the system posits a $+\mathbf{s}$ event if the consonant is before the vowel, and a $-\mathbf{s}$ event if the consonant follows the vowel.

- Obstruent consonants generate posited $\pm\mathbf{c}$ events: for every obstruent consonant a $+\mathbf{c}$ is posited at the beginning and a $-\mathbf{c}$ at the end. However, in the case of obstruents adjacent to other obstruents, these events are considered non-required.

This means that there will be a $-\mathbf{c}$ at a stop closure following a fricative or other obstruent consonant, and a $-\mathbf{v}$ at a stop closure following a sonorant segment.

Note that these rules often result in two events ($+\mathbf{c}$ and $-\mathbf{v}$, or $+\mathbf{v}$ and $-\mathbf{c}$) co-occurring at a boundary between two segments with no clear order. For example, either order is possible between the $+\mathbf{c}$ and $-\mathbf{v}$ events at a boundary where a vowel is followed by a fricative, though there is some correlation with voicing as there is more likely to be overlap between the segments if the fricative is voiced. This problem required modifications to the scoring algorithm for increased flexibility, as discussed below. There are also some special cases:

- An extra non-required $-\mathbf{c}$ event is inserted halfway through a stop burst, as many bursts (particularly those for unvoiced stops) will have an offset of energy both at the point where the fricated noise source stops and aspiration begins, and again at the end of the aspiration entering a following segment.

- An extra non-required $+\mathbf{c}$ event is inserted after the first one for a velar (/k/ or /g/) stop burst, because these stops often generate a double burst on release

Table 3.2: Dynamic Programming (DP) cost structure

Type	Cost
Match	1/ms distance
Insertion	50
Deletion	50
Deletion (non-required event)	0
Substitution (same polarity)	50 + 1/ms distance
Substitution (opp. polarity)	100 + 1/ms distance

due to the larger area of the closure region between the tongue dorsum and the palate.

- A non-required $\pm$**s** event pair is generated between adjacent vowels or adjacent sonorant consonants, as it is expected that some energy change may occur at these points and this allows them to be matched with a posited event.

3.4 Scoring

A standard algorithm used for scoring speech recognizer performance at the phonetic level was modified to support scoring landmark results. The algorithm was derived from the DARPA speech recognizer performance evaluation tools [32], a standard set of code for scoring results of speech recognition systems. This code aligns a recognized token string with reference labels using a dynamic programming algorithm. The original code supported scoring costs for insertions, deletions, and substitutions in a stream of labels. Modifications were made to perform the task of landmark scoring: a) a cost was added for the difference in time (in ms) from the posited label (as per Section 3.3) to the detected label, to ensure that label matches and

substitutions were close in time (insertion/deletion costs are equivalent to the cost of a matching label off by 50ms); b) support for non-required events with zero deletion cost was added; c) support for pairs of co-occurring events which could be found in either order was added, for example the onset of a fricative at the same point as the offset of the preceding vowel; and d) substitution cost was doubled in the case that the polarity was incorrect, such that $+\mathbf{c}$ for $-\mathbf{c}$ was a more costly substitution than $-\mathbf{v}$ for $-\mathbf{c}$, as it was more likely in the polarity mismatch cases that there was actually both an insertion and a deletion, rather than just a substitution. Due to inclusion of a cost for the distance in time between the posited and generated events, this type of substitution would never be chosen by the scoring algorithm, as the cost structure makes it cheaper for the system to count it as an insertion plus a deletion.

Additional adjustments in the final score were made to ignore insertions before the beginning and after the end of the labelled speech, under the assumption that integrating an endpoint detector in the system would prevent positing events at these locations. A sample of the results of the scoring process over an utterance is shown in Table 3.3 for the sentence "A huge power outage rarely occurs." (the same utterance shown in Figure 1.2), spoken by a female speaker. Note that the deletions only occurred for sonorant $\pm\mathbf{s}$ events at times 356.0ms, 1607.44ms, and 1712.75ms.

Table 3.3: Sample scoring results. A * in the recognized column indicates a deletion; a * in the reference (posited event) column indicates an insertion (with the label of the region in which it was inserted). A question mark (?) in the reference label indicates a non-required event.

Time	Recog.	Ref.	Label(s)	Time	Recog.	Ref.	Label(s)
145.00	+c	*	h#	1184.00	-s	-s	aw;dx
155.00	+v	+v	h#;ax	1209.00	+s	+s	dx;ix
190.00	-c	*	ax	1282.00	-v	-v	ix;dcl
207.50	+c	+c	ax;hv	1323.00	+c	+c	dcl;jh
210.00	-v	-v	ax;hv	1384.00	-c	-c	jh;r
310.00	+v	+v	hv;y	1445.00	-c	*	r
310.00	-c	-c	hv;y	1467.50	+v	+v	jh;r
356.00	*	+s	y;ux	1479.00	+s	+s	r;eh
483.00	-v	-v	ux;dcl	1607.44	*	-s	eh;r
514.00	+c	+c	dcl;jh	1663.12	*	+s?,-s?	r;l
606.00	-c	-c	jh;pcl	1712.75	*	+s	l;iy
657.00	+c	+c	pcl;p	1855.00	-v	-v	ix;kcl
664.00	-c	-c?	pcl;p	1903.00	+c	+c	kcl;k
682.00	+c	*	p	1957.00	-c	-c?	kcl;k
707.50	+v	+v	p;aw	1940.91	*	+c?	kcl;k
720.00	-c	-c?	p;aw	1992.00	+v	+v	k;er
978.00	-v	-v	axr;q	1992.50	-c	-c?	k;er
1007.00	+c	+c	axr;q	2212.00	-v	-v	er;z
1025.00	+v	+v	q;aw	2233.00	+c	+c	er;z
1037.00	+s	*	q	2379.00	-c	-c	z;h#
1060.00	-c	-c?	q;aw	2493.00	-c	*	h#

Insertions:	4	(+ 2 beyond endpoints)
Deletions:	3	(+ 3 neutral deletions of non-required labels)
Substitutions:	0	
Net errors:	7	(+ 5 accounted for)

Chapter 4

PROCEDURE

This chapter describes details of the implementation and testing of the temporal event extraction system. Section 4.1 describes the database used for training and test; Section 4.2 discusses the parameter training procedure; and Section 4.3 is a list of the tools used in implementing the system (with references to code in the Appendix).

4.1 Database

The TIMIT database [33] was used as a corpora of labelled speech data. This is a widely available database of speech labelled at the level of words and phoneme strings. Although it would have been more useful to use a database labelled at the landmark level (e.g. a database currently under development at the Massachusetts Institute of Technology [34]), a large enough database of this type was not yet available.

The TIMIT database consists of 6300 utterances spoken by 630 speakers, of which 4620 utterances make up the suggested training set and 1680 are in the test set. These include three types of sentences:

- **sx** (5 sentences per speaker) — phonetically compact sentences, a set of sentences designed to have a full set of phonetic contexts, in terms of co-occurring pairs of phones

- **si** (3 unique sentences per speaker) — phonetically diverse sentences, a set of sentences from existing text (including the Brown corpus)

- **sa** (2 sentences, spoken by every speaker) — "shibboleth" sentences designed to expose interesting variations across dialect types (not useful for this task)

Training was performed using a set of 20 of the **sx** sentences (spoken by 10 males, 10 females) randomly drawn from the TIMIT training set. Testing was performed using all 120 **sx** sentences from a subset of the TIMIT test set referred to as the TIMIT core test set, which (including an additional 3 **si** sentences per speaker) contains a well balanced and complete set of phonetic contexts and includes 40 **sx** (and 24 **si**) sentences spoken by 8 female speakers, and 80 **sx** (and 48 **si**) sentences spoken by 16 male speakers evenly distributed over all 8 dialect regions.

4.2 Training procedure

Some adjustment and training was performed on a number of the time, energy, and confidence level thresholds involved in pitch detector interpretation and event extraction.

The training procedure involved adjusting a set of 12 parameters, listed in Table 4.1. The procedure was a somewhat ad hoc series of adjustment of sets of thresholds considered to be related (the subgroups noted in the table). Each set was trained over a matrix of specified values to maximize the score:

$$S_{\text{opt}} = (\#\ \textit{req'd matches}) - (\#\ \textit{insertions}) \tag{4.1}$$

This is equivalent to minimizing the total error rate, as the base number of posited required events will not change. This training process was iterated twice to ensure some degree of convergence. Although minor improvements were made by adjusting the thresholds, it is likely that they are too closely associated with particulars of the database (recording characteristics, etc.) for general use. Note on the other hand that in a number of cases, there were parameters (individual or pairs) that could be chosen at a number of values with either no change or very minimal change

Table 4.1: Parameters with trained values. Pon and Poff refer to the boundary 'events' of a periodic region; APon and APoff are the corresponding locations for an aperiodic region.

Parameter	Description	Value
Sil_thres	Silence threshold	-75.0dB
Pon_before	Maximum time between Pon and corresponding onset peak, if peak *precedes* Pon	20.0ms
Pon_after	Maximum time between Pon and corresponding onset peak, if peak *follows* Pon	5.0ms
PER_thres_RGN	"Region threshold" on P_{conf} to consider a region as periodic	10.0
PER_thres	"Boundary threshold" on P_{conf} to located ends of a periodic region	5.0
Poff_time	Maximum time between Poff and corresponding offset peak.	45.0ms
AP_time	Maximum time between APon/APoff and corresponding onset/offset peak	30.0ms
APER_thres_RGN	"Region threshold" on AP_{conf} to consider a region as aperiodic	12.0
APER_thres	"Boundary threshold" on AP_{conf} to located ends of an aperiodic region	6.0
On_peak_thres	Minimum peak height in onset measure	5.0
On_dip_thres	Minimum dip between peaks in onset measure	3.0
Off_peak_thres	Minimum peak height in offset measure	4.0
Off_dip_thres	Minimum dip between peaks in offset measure	3.5

in performance according to the metric used, which implies a degree of generality. For example, when training the thresholds for aperiodic regions, there were a number of threshold pairs for APER_thres_RGN and APER_thres (for example, APER_thres_RGN at 10.0 and APER_thres at 8.0) that caused no significant change in the results. It is likely that a real system would use higher-level processes to adjust some of these thresholds dynamically. Note Sections 5.1 and 7.1 which discuss ways that the pitch detection algorithm could be improved to minimize dependency on thresholding.

4.3 Implementation

Several stages were involved in running the system. Initial data processing consisted of the auditory filter bank, which is a C program modified slightly from a version acquired from Michael Heinz and Laurel Carney [28].

The Hilbert envelope operation was computed using an FIR approximation in Matlab^{TM} (although a higher performance version was developed). All other components described in Chapter 2 regarding the low-level detector (first difference, onset/offset detector, and periodicity analysis) have been implemented in Matlab, with some glue code written in Perl or as BASH shell scripts. This includes the event detector, which is written in Matlab as well.

For scoring there are C++ programs for the positing of events from TIMIT labels, and the matcher (described in Section 3.4, and derived from DARPA code [32]). Again, there is a large amount of scripting involved in this process. Final analysis of the data is mostly written in Perl.

Much of this code is available in the Appendices.

Chapter 5

RESULTS

The system works reasonably well for extracting events that are strongly present and, for the purpose of scoring, consistently labeled. The pitch detector is a critical component, in particular regarding the temporal accuracy of the boundaries of regions of periodic and aperiodic excitation; analysis of the performance of the periodicity detector is covered in Section 5.1. Overall, 70.8% of events were detected including 87.4% of a set considered to be robust for temporal information (77.2% and 90.7% on the training set), including over 90.0% for particular categories. Detailed performance results are discussed in Section 5.2, including the metrics used and analysis of the major sources of error.

5.1 Periodicity detector performance

In order to optimally locate events that are indicated by the boundaries of regions of periodic or aperiodic excitation, the periodicity detection system must perform well. In particular, for accurate event detection, temporal accuracy of these region boundaries must be acceptable—this is part of the reason for the various trained time thresholds for integration of information from the two low-level detectors. The modified two-level threshold used to detect boundaries was designed with this goal in mind (though there may be further improvements that could be made).

Performance of the pitch detector was compared with the commercially available Entropic ESPS [35] **get_f0** component as a reference. The voicing decisions agreed 88.7% of the time on the test set (89.1% on the training set). The temporal detector

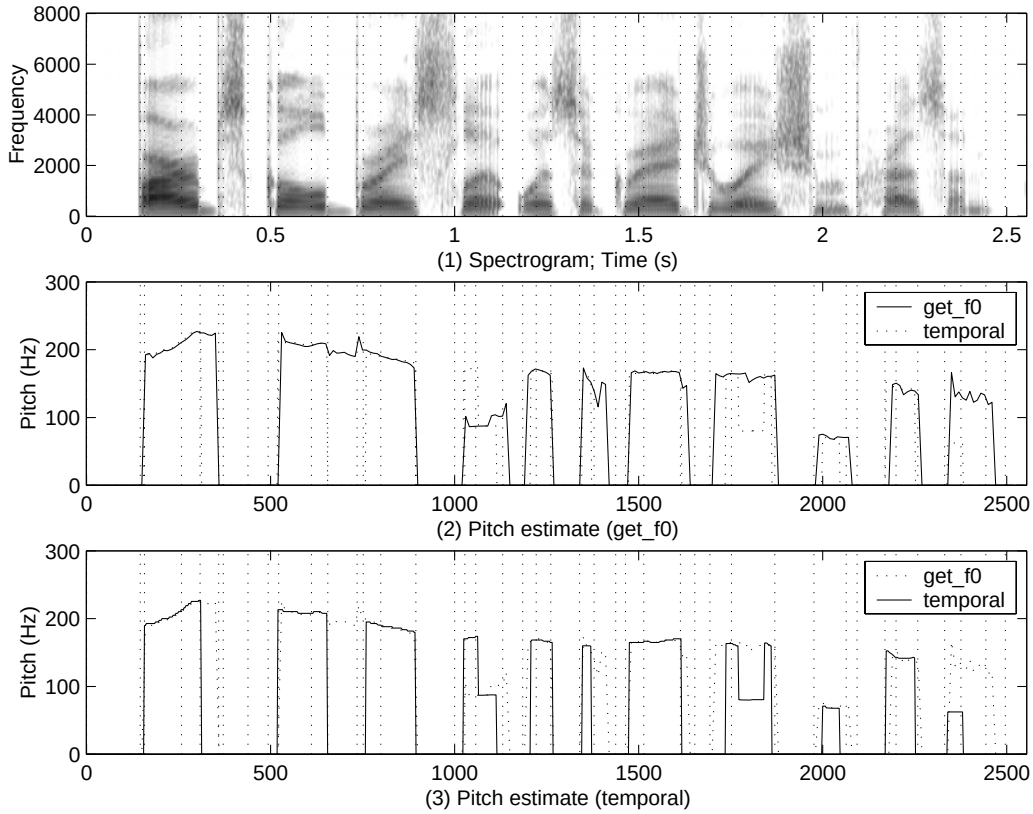

Figure 5.1: Sample of pitch analysis, compared with reference **get_f0** detector. The utterance is the sentence "Barb's gold bracelet was a graduation present," spoken by a female speaker.

is somewhat more conservative in deciding that a region is periodic; for example the voice-bar region near 700ms in Figure 5.1, which is labeled as aperiodic by the detector. Error in pitch decisions, scaled relative to median pitch for each utterance, was 13.4% (12.0% training).

It was noted that a major type of error in both pitch detectors was occasional choice of a pitch of approximately one half the correct pitch (twice the correct pitch period). One example of this type of error is located at approximately 1800ms in Figure 5.1. This type of error is due to the time-domain peak picking used in the temporal detector system (**get_f0** also involves some time-domain methods). This

error occurs more often in female speech because the male pitch period lengths are often above one half the 20ms window length used (if pitch $\leq$ 100Hz), implicitly preventing this type of error. In order to determine how often this factor-of-2 error occurred, frames were counted in which the error between the two detectors was improved by multiplying one or the other by a factor of 2. For the test set, this occurred in 0.55% (0.91% for female speech, 0.37% for male speech) of the frames for the **get_f0** detector, and in 1.62% (2.78% for female speech, 1.03% for male speech) of the frames for the temporal detector. Performing the adjustments reduced pitch error to 8.58% on the test set, concentrated in the male speech (and only 4.91% on the training set).

5.2 Event detection performance

A set of summary statistics was defined to analyze matching results. All are defined in terms of the base rate T_R, the number of posited tokens not counting neutral deletions (of tokens marked as non-required). Defining T_P as the total number of posited tokens, D as the number of error deletions (of required tokens), D_N as the number of neutral deletions, S as the number of substitutions, and I as the number of insertions, the metrics are computed according to the following formulas:

$$\text{base token count:} \quad T_R = T_P - D_N \quad \textit{(base rate of matched tokens)} \tag{5.1}$$

$$\text{detection rate:} \quad R_M = \frac{T_R - D - S}{T_R} \quad \textit{(also called 'match rate')} \tag{5.2}$$

$$\text{deletion rate:} \quad R_D = \frac{D}{T_R} \tag{5.3}$$

$$\text{substitution rate:} \quad R_S = \frac{S}{T_R} \tag{5.4}$$

(collectively, deletion and substitution are referred to as 'miss rate')

$$\text{insertion rate:} \quad R_I = \frac{I}{T_R} \tag{5.5}$$

Table 5.1: Match rates, strongly expected event types

Event type	Train	Test
closure preceding stop consonant	88.5%	89.6%
stop burst	92.2%	88.3%
voicing onset following stop	88.4%	82.5%
africate	100.0%	93.4%
africate – voicing	88.9%	91.4%
strident fricative	93.5%	90.3%
strident fricative – voicing	80.9%	82.8%
Summary	89.6%	87.1%

Events were detected with overall detection rate of 70.8% on the test data set (77.2% on the training set), and an insertion rate of 12.0% (10.1%). Nearly half of the error rate was due to missed boundaries between sonorant consonants and vowels, an event type that was detected with only 41.6% (46.7%) accuracy. Note that even in the transcription of the TIMIT database [33], an explicit rule was used to insert these boundaries when they were not reliably locatable by a trained human transcriber. The other major error source was from weak (non-strident) fricatives, for a total of over 60% of misses from these two sources. Not counting the two major sources of error (see Table 5.2), the detection rate was 83.2% (87.9%); and the detection rate for a set of the most robustly expected event types was 87.1% (89.6%). Most of the insertion rate was due to fluctuations in vowel, fricative or closure regions.

Among robust event types that performed well were stop consonants and strident fricatives, with an 89.6% match rate for stop closures, 88.3% for stop bursts, and 90.3% for strident fricative consonantal events for the test set. Even better results

were achieved for africate consonants (/ch/, /j/), with 93.4% match rate for the consonantal events and 91.4% for corresponding voicing events (though note that these were relatively infrequent segment types). For stop bursts, almost half of the errors (46.4%, 6/13 of the deletions and 20/43 of the substitutions) were due to /b/ bursts which tend to be relatively weak or hard to detect (discussed below). Also, for strident fricatives a large proportion of deletions were due to the voiced /z/ and /zh/ fricatives which can have a very difficult to locate boundary due to overlap with neigboring segments; specifically, these stridents accounted for 76% of the $\pm$**c** (consonantal) deletions and 64.0% of the $\pm$**v** (voicing) deletions among strident fricatives.

Regarding boundaries between segments with the same manner, as mentioned in Chapter 3, there were detections of events at 21 of 28 (75.0%) fricative-fricative boundaries and 15 of 47 (31.9%) sonorant-sonorant boundaries (most likely located at nasal-glide or vice versa) detected in the test set, and 5/7 (71.4%) and 2/7 (28.6%) respectively in the training set. More results for robustly expected event types are listed in Table 5.1, and detailed results for all common event types are detailed in Tables 5.3 and 5.4.

5.2.1 Major sources of error

There were a number of error types that were responsible for the majority of the error rate. These include the following:

Sonorant events The most difficult type of event to locate from strictly temporal information, accounting for approximately 40% of the miss rate, were those associated with sonorant consonants. In previous work on abrupt landmark detection by Liu [12], slightly better performance was obtained by adding some dependency on spectral structure (in particular, sonorant events were associated with energy change in midrange channels from 800-5000 Hz). However, it has been noted in previous

Table 5.2: Summary of results: Major sources of error (% of misses/insertions)

Misses: Context	Train	Test
vowel-sonorant	42.0%	39.9%
weak fricative	20.3%	22.9%
stop events	15.6%	13.8%
closures	*7.1%*	*4.5%*
bursts	*3.9%*	*3.8%*
voicing onsets	*4.7%*	*5.3%*
strident fricative	8.8%	9.4%

Insertions: Context	Train	Test
stop closure	24.5%	20.2%
vowel	13.8%	28.0%
fricative	25.5%	23.2%
strident fricative	*12.8%*	*12.1%*
weak fricative	*12.8%*	*11.2%*
stop	9.6%	9.2%
sonorant consonant	9.6%	8.7%

work by Espy-Wilson [9, 10] that it is critical to include formant tracking to locate these events, as the major cue for sonorant consonants is a decrease in the first formant frequency in conjunction with a sharp change in the frequency of the 2nd (/w/, /y/) or 3rd (/r/, /l/) formant. In a strictly temporal system (or when spectral information is unavailable), it would be necessary to posit likely insertion—especially in pre-vocalic context—based on features such as vowel duration, that is positing a potential missed sonorant consonant if a vowel is too long according to a high-level duration model[1]. At these points, further analysis such as formant tracking would be required to clarify the existence of a landmark.

Weak fricatives Another difficult to locate event class were those that signal the existence of a weak (or non-strident) fricative, which include /f/, /v/, /th/ (as in 'thin'), and /dh/ (as in 'this'). In some cases, particularly for the voiced fricative /v/, it is likely that the fricative has actually been produced as a sonorant. In other cases the events associated with these segments are simply not very strong.

[1] though there may be other reasons for a long vowel, such as a pair of adjacent vowels

Table 5.3: Detailed results (training set)

	Match		Deletion		Substitution		Total
Total tokens	718	(57.6%)	345[a]	(27.7%)	50	(4.0%)	1246
Counted tokens (T_R)	718	(77.2%)	162	(17.4%)	50	(5.4%)	930
Stop closures	115	(88.5%)	2	(1.5%)	13	(10.0%)	130
Stop bursts	95	(92.2%)	0	(0.0%)	8	(7.8%)	103
Stop Von	76	(88.4%)	10	(11.6%)	0	(0.0%)	86
Africates	24	(100.0%)	0	(0.0%)	0	(0.0%)	24
Africate Von	8	(88.9%)	1	(11.1%)	0	(0.0%)	9
Strident fr	100	(93.5%)	5	(4.7%)	2	(1.9%)	107
Str fr voic	55	(80.9%)	9	(13.2%)	4	(5.9%)	68
Summary robust	447	(89.6%)	27	(5.4%)	25	(5.0%)	499
Weak fric	31	(56.4%)	23	(41.8%)	1	(1.8%)	55
Wk fr voic	30	(61.2%)	16	(32.7%)	3	(6.1%)	49
Sonorants	78	(46.7%)	77	(46.1%)	12	(7.2%)	167
Fricative – fricative	5	(50.0%)	5	(50.0%)	0	(0.0%)	10
Sonorant – sonorant	2	(12.5%)	14	(87.5%)	0	(0.0%)	16
Other	182	(86.3%)	20	(9.5%)	9	(4.3%)	211

Note that there were 651 (75.4%) matches out of 863 required tokens.

133 (10.7%) total insertions.

94 (10.1%) counted insertions.

[a] includes 183 neutral deletions

Table 5.4: Detailed results (test set)

	Match		Deletion		Substitution		Total
Total tokens	3467	(51.9%)	2046	(30.7%)[a]	276	(4.1%)	6674
Counted tokens (T_R)	3467	(70.8%)	1155	(23.6%)	276	(5.6%)	4898
Stop closures	561	(89.6%)	25	(4.0%)	40	(6.4%)	626
Stop bursts	422	(88.3%)	13	(2.7%)	43	(9.0%)	478
Stop Von	358	(82.5%)	73	(16.8%)	3	(0.7%)	434
Africates	99	(93.4%)	3	(2.8%)	4	(3.8%)	106
Africate Von	32	(91.4%)	1	(2.9%)	2	(5.7%)	35
Strident fr	537	(90.3%)	55	(9.2%)	3	(0.5%)	595
Str fr voic	327	(82.8%)	50	(12.7%)	18	(4.6%)	395
Summary robust	2219	(87.1%)	217	(8.5%)	111	(4.4%)	2547
Weak fric	218	(52.9%)	183	(44.4%)	11	(2.7%)	412
Wk fr voic	216	(61.7%)	122	(34.9%)	12	(3.4%)	350
Sonorants	407	(41.6%)	497	(50.8%)	74	(7.6%)	978
Fricative – fricative	21	(60.0%)	14	(40.0%)	0	(0.0%)	35
Sonorant – sonorant	15	(15.3%)	83	(84.7%)	0	(0.0%)	98
Other	767	(78.7%)	140	(14.4%)	68	(7.0%)	975

Note that there were 3346 (70.0%) matches out of 4777 required tokens.

885 (13.3%) total insertions.

588 (12.0%) counted insertions.

[a] includes 891 neutral deletions

The most commonly deleted events were on the release of the consonant /dh/, 49 deletions (of both **−c** and **+v**) in the case of a following reduced vowel (such as in the words 'this' or 'the'), and 33 when followed by some other vowel. In addition to the 67 events deleted on the release of a /v/, these account for half of the weak fricative deletions (which also include deletions on closures for voiced and unvoiced segments, and releases on unvoiced weak fricatives). More research is required into the robust location of these events using temporal parameters.

Voiced stops The stop bursts for voiced stops can be very close to voicing onset, and therefore it can be hard to separate the two distinct events: the onset of the stop burst, and the following onset of glottal vibration (especially considering the threshold mechanism used for integration of low-level events). It may be possible to clean up some of these cases by improvement of the temporal accuracy of periodic regions. This location in stop consonants is the critical region for further tuning of the periodicity detector, as it is the major area where subtle changes to the detector can improve performance, where temporal information is available in the signal (unlike the case of sonorant consonants). However, it may also be necessary to expand the architecture to cope with certain cases: note for example the /b/ at 700ms in Figure 2.7a, in which the onset peak for the /b/ burst and beginning of periodic excitation (vowel onset) which follows are exactly matched to the labels, but very close together. As such, there is no clear way to make a hard decision that they are not the same event in the current architecture, which uses thresholds in time between the two types of low-level events to take into account the variability in time of onset/offset peaks that correspond to boundaries of periodic regions. This suggests a modification of the system to compute probabilities of a number of different alternatives, and track these multiple paths in higher level processing. The issue of locating boundaries of periodic regions with higher temporal accuracy is discussed in more detail below.

More specifically, the real issue is not necessarily location of every event in time so much as the three-way distinction between a) an unvoiced aspirated stop, with a significant (>50ms) aperiodic excitation preceding a vowel onset; b) a voiced stop with a very short (<30ms) aperiodic excitation, which may coexist with low-level periodic excitation throughout the closure region (known as voice-bar); and c) a simple vowel onset, which may begin with an irregular (in time) excitation known as glottalization — which can be detected because it will be spectrally very similar to the following vowel. The distinction between the last two cases may be served best by a spectral comparison between the initial irregular burst of energy and following regular excitation. Note that the most common substitution was a voiced stop burst recognized as a $+\mathbf{v}$ event (33 of the 43 substitutions for stop bursts; in every case, the expected $+\mathbf{v}$ also counted as a deletion). There were also a number of cases of recognition of a voiced stop burst as a $+\mathbf{s}$ event (5 more substitutions) or where it was simply deleted (6 instances).

There are also a number of misclassified closures: 23 stop closures following reduced vowels (e.g. schwa) which were recognized as $-\mathbf{c}$ rather than $-\mathbf{v}$, which implies that the voiced region may not have been located at all. In voiced stops there were also a small number (7) of cases where low-level periodic excitation in the closure region caused some $-\mathbf{v}$ events to be misclassified as $-\mathbf{s}$.

Insertions The major insertion locations were within vowels and stop closure regions, particularly the 28.0% (20.2% training) of insertions in labelled vowel segments, corresponding to an insertion rate of 6.8% (5.3%) for those segments. Although the absolute number of insertions in vowel regions is high, it's not surprising considering that roughly $\frac{1}{3}$ of the labelled segments are vowels (33.0% training, 30.8% test), especially noting the high degree of fluctuation in vowel regions due to voicing. Another 20.2% (24.5%) of the insertion rate occurred in closure regions, corresponding to an insertion rate 12.1% (19.2%) of the labelled closures. This was

particularly high in closures for /k/ consonants (31.4%, 27/86 cases). The closure results are due in large part to the increased sensitivity of the difference operator in these regions, and could possibly be improved by adjusting the tuning of the short difference time used in silent regions for the onset/offset detector.

Also note the insertion rate in fricative regions, which reached 12.2% (20.0%) for strident fricatives, and 16.9% (32.4%) for weak fricatives. It seems that more work will be required to determine the correct smoothing algorithm or threshold adjustments in these regions. It may be the case that increasing onset/offset peak thresholds in response to knowledge about lack of periodicity would be a useful modification. Additionally, increasing the window length even further (above 30ms) in these regions may help to address this.

5.2.2 General issues

Database Issues One critical issue with the database under study is that the hand-transcribed labels aren't always strictly based on acoustics. For example, in the rules used to generate the TIMIT transcriptions [33], if there was no clear acoustic boundary between a semivowel and an adjacent vowel to the trained human labeler, then 1/3 of the sonorant region was labeled as the semivowel. Another lack of specificity in the database is related to variability in mode of production due to non-canonical realization of certain segments (such as the sonorant /v/ discussed above). In this case the issue is that although the acoustics for two versions of a segment are categorically different, and therefore there are a different set of landmarks acoustically realized for that segment, the two segments are nevertheless labeled with the same token. Both types of problems may be addressed by use of a more explicitly landmark-oriented database [34].

Resolution A general concept that may be an issue is temporal resolution in the system. The tuning of the difference time may cause the detector to be less sensitive

to some types of events in certain contexts. For example, the particular difference time in sonorant regions may not be long enough to detect the slow transitions in those regions. Another issue of temporal resolution is the location of boundaries of periodic and aperiodic regions, in the periodicity detection component. This has resulted in a large number of threshold parameters in time and confidence level that have required explicit training which likely gives results that depend upon the details of signal conditioning and perhaps speakers in the database. It would be preferable to develop dynamic thresholds (perhaps based on more detailed auditory models).

Regarding the tuning of difference time for the onset/offset detector, there are a number of cases where there may be problems in the sensitivity level in some regions. One example is that the existence of a voice-bar in voiced closure regions may decrease temporal resolution in those channels, relative to channels in which there is silence — though possibly this is an appropriate response and therefore a good model. Other examples include both of the insertion problems noted above, in closures and fricative regions. Two possibilities that exist for improvement of this would be either a) to modify the algorithm used to adapt the difference length, which could perhaps involve using cross-spectral information in adjusting difference lengths (currently adaptation is based only on the individual channel AMDF results); or b) use of multiple resolutions concurrently. Note in particular studies by van der Horst et al. [26] which show that filtering out particular excitation frequencies from temporal envelopes are related to different types of cues.

In the pitch detection subsystem, although the two-level threshold system is an improvement over a simple threshold, what is more generally required is to find a 'kink' in the function (e.g. a measure based on finding a zero in the 2nd derivative). It may also be helpful to implement a dual-level system ('low-level' periodicity such as voice-bar vs. 'high-level' periodicity in e.g. vowels). Note that long-term integration to generate a model of the expected range of the pitch period for a particular speaker would aid in confidence measures in the 'low-level' periodicity voice-bar regions when

only a small number of channels show periodic excitation (or channels that exhibit only weak periodicity, very low energy).

Chapter 6

CONCLUSIONS

This work has shown that use of temporal information for landmark detection is feasible, particularly for a subset of robust abrupt events such as stop bursts. Although previous studies have investigated the use of temporal information in particular cases or as an additional measure, this work extends this body of work by using temporal information everywhere as the primary information source. It has also pointed to certain areas where spectral features and perhaps more subtle temporal features (on a longer time scale) are important, particularly for landmarks related to sonorant consonants. As noted by use of a tunable onset/offset detector it was determined that some locations require different degrees of sensitivity to temporal information.

The present implementation performs reasonably well. However, there are a number of key areas where accuracy could be improved, particularly in use of prediction and longer term integration of information. These include use of more contextual information (with the potential for decreased computation) in the pitch detector, improved adaptation of the sensitivity of the onset/offset event detector, and modification of the system to dynamically adapt thresholds based on the signal.

According to the motivations discussion in the introductory chapters, it is clear that temporal information is used by the human speech recognition system, and so should be critical to achieving high quality in a computer speech recognition system in all conditions. To further develop this work, and combine temporal information with additional (spectral) feature types, it will be important to rigorously test a particular set of algorithms (such as the one described in this thesis) to answer two

critical questions: 1) whether integrating temporal information with a spectral-based system improves performance of the recognizer; and 2) whether such a recognizer (temporal or combined) is more resistant to noise than a primarily spectral feature-based system.

It will also be important to develop higher-level structure of a knowledge-based recognizer and to incorporate any higher level temporal features at that level (such as duration modeling as studied in [36]).

Chapter 7

AREAS FOR FURTHER WORK

There are a number of directions in which this work can be developed. First, further analysis of the specific parameters—possibly including more detailed analysis of perceptual studies—could improve performance in extraction of temporal information. Next, it will be important to integrate the temporal detection system with other types of signal analysis based on spectral parameters to develop a more robust event detection system. It will be critical, however, to integrate with the higher-level structure of a recognition system, to generate output at the phonemic or lexical levels. Finally, it will be important at some stage to consider the issues involved in use of the parameters in a usable (near real-time) automatic speech recognition system.

7.1 Improving detector

There are several ways that the temporal signal analysis system can be improved upon. The primary areas that seem to be important include addressing issues of more dynamic adjustment of the system (rather than the current set of explicit thresholds), generation of multiple hypotheses, and improving accuracy of the pitch detector in a couple of key areas.

Constant thresholds are not the optimal algorithm for adjustment of the system to varied signal conditions, even over the course of a single utterance. It is likely that adjusting the system based on more static constraints such as phoneme rate (decreasing thresholds in regions where the system isn't detecting many events) could improve accuracy. Another option might be some degree of reanalysis of regions to

locate more subtle events (smaller peaks) at a later point in time if there is evidence that something was missed. Another part of the adaptation problem is to study and improve the algorithm used to adapt the difference operator, possibly including some degree of cross-channel information. However, it is important not to overgeneralize the problem: any clear limitations on human perception, for example, should be used to constrain the types or rates of adaptation.

Another way to improve the system in situations where there is a lot of variability or uncertainty is to delay hard decisions by entertaining a number of hypotheses. This allows a slight increase in overgeneration of events (insertions) which can be clarified at a later point in the system, possibly based on high-level information. This will be especially critical in cases where some information has been degraded such that the system may be able to be confident only of the fact that it doesn't have enough information to make a decision based only on local signal content.

Periodicity detection algorithms The periodicity detection routines in particular have a number of areas where they could be improved. The periodicity detection system is a major component in the system that has undergone a significant degree of tuning and improvement. The current implementation requires a large amount of computation, but is approaching the accuracy of a relatively robust comparison system. Long-term integration of information could improve both accuracy and performance. There is also work to be done in temporal accuracy of periodic regions for the task under study, which is a somewhat more stringent requirement in this system than required for most other applications of pitch detection algorithms.

Regarding precise temporal accuracy, attention needs to be paid to the methods used to locate boundaries of regions, in particular for the onset of periodicity. The current system extended the threshold concept for periodicity confidence values to a high "region threshold" and a lower "boundary threshold", but it seems that applying a more complex algorithm to this boundary location problem may be helpful.

One example might be to use a zero-crossing in the 2nd derivative of the confidence function, finding a sharp kink in this 'signal' to locate a more accurate boundary.

Another part of the problem, however, is to consider periodicity of regions with only a small amount of periodic excitation, for example voice-bars in the closure region of a voiced stop consonant where most of the periodic excitation has relatively low energy and is concentrated in low frequency regions. In this case, it might help to incorporate additional information when computing the confidence measure: if there aren't very many confident estimates, but they are consistent with recent pitch estimates, this could be support for such a low-level periodic excitation. It should be possible to incorporate a simple model of the general range of pitch expected as well as some limits on rate of change. This may also improve performance by allowing use of fewer channels, and/or examining smaller ranges of the AMDF function.

In terms of decreasing computation, it may also be worthwhile to consider use of a simple interval histogram (between raw peaks or dips in the envelope signals, or some other representation) directly rather than computing the AMDF. This could also involve use of a more complete auditory model (such as neural spike generation), which would lead to the use of a more accurate auditorily motivated model for pitch perception based on an interspike interval histogram [23], which was in some sense the inspiration for histogram processing in this thesis. However, it is important to weigh the benefits of various parts of such a model against computational cost.

7.2 Integration with spectral information

Another important area of development will be combining with different types of information, such as spectral information. There are a number of very specific ways in which spectral information can be used, for example analysis of harmonic structure to improve pitch detection, and comparison of spectra to distinguish stop bursts from glottalization. More generally, it will be important to know which

parts of the spectrum can be trusted in noisy or degraded environments. Adding spectral information could involve incorporating a formant tracker, or other types of modeling of spectral structure of different segment types. One important aspect will be determining how early in the processing chain different types of cues should be considered; for example, rough formant tracking (especially of the first formant) might be helpful in locating sonorant landmarks, but it is an expensive and possibly unreliable task to perform everywhere.

7.3 Completing the system

Higher-level processing needs to be developed, the first stage of which would most likely be a segment-level representation. Following this, higher level stages for lexical access (i.e. word recognition), and then language or dialogue modeling would be necessary for a complete system. It is important to keep in mind the use of temporal information at all stages in the system. At the segment and words levels of representation (and perhaps higher levels if prosodic information is considered), a system could incorporate use of duration cues for full use of temporal information. For example, at the segment level this may work well in a prediction-oriented system where detected landmark times are translated into posited segment durations which can be tested against a model (e.g. as in [36]). This information could be used to rank multiple hypotheses, and to determine which regions may contain extra landmarks that were not detected in early stages and need to be inserted, which may involve scheduling more detailed signal analysis.

7.4 Development of a real-time system

Finally, it is critical in building a functional speech recognition system that it be able to run near real-time. This requires at a theoretical level that the computation not depend on too much future information to posit events: a rigorous study of

these issues has not been undertaken, though the maximum forward information requirement of the current system does not seem to be very large. However, increased attention to these issues could improve the delay, especially important as the landmark detector is only the first stage of analysis in a full system.

There is also a required degree of efficiency in processing. Most of this simply requires implementation in a language other than Matlab, and possibly optimization of the filterbank code based on necessary resolution. However, there are also a number of ways in which this system can be simplified for this purpose by removing unnecessary computation. Further research is required in this area, but some possibilities may include paying more attention to the strongest channels (perhaps within particular ranges), a degree of pitch hysteresis to minimize AMDF computation (related to ideas of prediction above). Some of these improvements could benefit from a study of how much information can be removed, but it is also important to identify which information is best to ignore: for the benefits of noise-tolerance it would be important to track information in those channels that seem to have good information about the speech signal rather than being obscured by noise.

Appendix A

AUDITORY FILTER BANK

Auditory filter bank implementation: `anmodheinz00.c` (Heinz et al., 2000)

```
/* anmodheinz00.C : Code for AN Model from Heinz et al. (2000) */

/* Cleaned up from Manuscript code 9/10/99 by M.G. Heinz
   (mgheinz@mit.edu) */
/* More cleanup done 3/1/00 to enable longer stimuli by A. Salomon
   (ariel@bu.edu, asalomon@alum.mit.edu) */
/* More cleanup done 5/1/00 for bin file output (saves disk space), by A. Salomon */

/* Basic multiple-fiber LINEAR, Human auditory-nerve model with all
   model-stage outputs saved to ascii files in the following format:

           DATA FILES set up as follows:
             line1: MAXCHS [time_vector]
             line2: Cf1 [ifr_Cf1]
             line3: Cf2 [ifr_Cf2]
             ...

   The variables: savestim, savebm, saveihc, saveifr can be set to 0
   to avoid saving the stimulus, basilar membrane, inner hair cell. or
   instantaneous firing rate, respectively.

   NOTE: set bin_save to 1 to save files in binary format (AS 5/00)

  */

/* The model and physiological responses of the model are described in
   the file: model_descript.ps, which is an excerpt from the
   manuscript.  Filter bandwidths are based on human psychophysical
   tuning curves, and DO NOT change with level (some code is left in
   this version that allows the filters to vary with level [set
   health=1], but the NONLINEAR version has not been tested.  Only the
   LINEAR version of the model was used in Heinz et al., 2000.

   Basic model response: Rsat=210 sp/sec, Rate Threshold=0 dB SPL,
   DR=20 dB, Max Onset rate = 850 sp/sec, max synch at LFs = 0.8,
   rolloff matches Johnson (1980). Synch Threshold = -13 dB. PSTs look
   good.  */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <math.h>
#include <time.h>

/*#define MAXTIMPTS 30000L   /* Maximum number of samples in time */
#define MAXCHS 100          /* Maximum number of frequency channels */

#define PIe   3.1415926536

#define DOFOR(i, to) for(i = 0; i < to; i++)

#define CMULTR(X,Y) ((X).x*(Y).x-(X).y*(Y).y)
#define CMULTI(X,Y) ((X).y*(Y).x+(X).x*(Y).y)
#define CTREAL(z,X,re) {(z).x=(X).x*(re);(z).y=(X).y*(re);}
#define CMULT(z,X,Y) {(z).x=CMULTR((X),(Y));(z).y=CMULTI((X),(Y));}
#define CADD(z,X,Y) {(z).x=(X).x+(Y).x;(z).y=(X).y+(Y).y;}

long stimtype,numchs,health,nstimpts;
long savestim,savebm,saveihc,saveifr, /* AS 5/00: */ bin_save;
long ichan,i,resp;
double xdum,Cf,delx,x,SPER;
double lowcf,xlowcf,highcf,xhighcf;

/*double stim[MAXTIMPTS],gtf[MAXTIMPTS],ihcL[MAXTIMPTS],ihc2[MAXTIMPTS];*/
/*double ifr[MAXTIMPTS];*/   /* channel x time */
double *stim, *gtf, *ihcL, *ihc2, *ifr;   /* AS 3/00 */
double anfCfs[MAXCHS]; /* chan: anf: ANFS */
double freq,phase,stimrft,levdbS,stimdur,Textra;

/* file pointers for writing to files */
FILE *fpstim,*fpbm,*fpihc,*fpifr;

void error(char *fmt, ...);
void stimulus();
void gamma4();
void synapse();
double erbGM(double);
double cmaph_x2f(double);
double cmaph_f2x(double);

struct complex { double x; double y;} ;
struct complex compexp(double);
struct complex compmult(double, struct complex);
struct complex compprod(struct complex, struct complex);
struct complex comp2sum(struct complex, struct complex);
double REAL(struct complex);
```

```
long round(double);

int main(void)
{

  /* &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& */
  printf("\n Heinz, Colburn, and Carney (2000) Auditory-Nerve Model (Linear, Human) -");
  printf("\n\tVersion 1.0 (9/10/99)\n");
  /*printf("This version reads in files with <= 30,000 pts. \n");*/

  savestim=1;     /* Boolean save variables: 0=NO, 1=YES */
  savebm=1;
  saveihc=0;
  saveifr=0;
  bin_save=1; /* AS 5/00 */
  /* set these flag to write out the output at different stage of the
     model to files for plotting (slows down program). */

  /* initialize variables */
  printf("Filterbank parameters: Set Center-filter freq [0], or Range of freqs [1]?: ");
  scanf("%ld",&resp); printf(" %d\n",resp);
  if(resp == 0)    {
    printf("CF of center filter in bank (Hz): ");scanf("%lf", &Cf); printf(" %f\n",Cf);
    printf("# of Filters in Bank: ");scanf("%ld", &numchs);  printf(" %d\n",numchs);
    printf("Distance between filters (mm along BM)- ");
    scanf("%lf", &delx);  printf(" %f\n",delx);
    /* NOTE: spacing between filters of 0.01mm ~= 1 HC spacing along BM */
    xlowcf=cmaph_f2x(Cf)-(numchs-1.)/2.*delx;
    xhighcf=xlowcf+(numchs-1)*delx;
    lowcf=round(cmaph_x2f(xlowcf));
    highcf=round(cmaph_x2f(xhighcf));
    printf("Resulting Filter Bank Center Frequency Range:  %.2f - %.2f Hz\n",
      lowcf, highcf);
  }
  else  {
    printf("Lowest frequency filter(Hz): ");scanf("%lf", &lowcf); printf(" %f\n",lowcf);
    printf("Highest frequency filter(Hz): ");scanf("%lf", &highcf);printf(" %f\n",highcf);
    printf("# of Filters in Bank: ");scanf("%ld", &numchs);  printf(" %d\n",numchs);
    if(numchs>MAXCHS) {
      printf("\n*** MAXCHS=%d in code. Need to increase!\n\n",MAXCHS);
      error("STOPPED");
    }
    xlowcf=cmaph_f2x(lowcf);
    xhighcf=cmaph_f2x(highcf);
    delx = (xhighcf-xlowcf)/(numchs-1);
    printf("Resulting delx = %f [Distance between filters (mm along BM)]\n",delx);
  }


  /* ***** CHANNEL CFs SETUP (anf) ***** */
```

```
  /* Set up ANF channels */
  xdum=xlowcf;
  anfCfs[0]=round(cmaph_x2f(xlowcf));
/*    printf("Channel %d = %4.2f Hz\n",ichan+1,anfCfs[0]);     */
  DOFOR(ichan,numchs-1) {
    xdum=xdum+delx;
    anfCfs[ichan+1]=round(cmaph_x2f(xdum));
/*      printf("Channel %d = %4.2f Hz\n",ichan+2,anfCfs[ichan+1]);     */
  }

  stimulus();


  /* ***** ALLOCATE MEMORY ***** (AS 3/00) */
  gtf = calloc(nstimpts, sizeof(double));
  ihcL = calloc(nstimpts, sizeof(double));
  ihc2 = calloc(nstimpts, sizeof(double));
  ifr = calloc(nstimpts, sizeof(double));
  if (!gtf || !ihcL || !ihc2 || !ifr)
    error ("Memory allocation error in main()");


  health=-1;
  /* (1=NL(healthy);
     0=impaired(LIN,broad,high threshold);
     -1=Previous Analytical Models(LIN,narrow,low threshold))
     -2=To isolate BW effect (LIN,broad,low threshold)) */
printf("Health [-1=LIN(sharp,Low Thresh); 1=NL; 0=LIN(broad,HT); -2:LIN(broad,LT)]: %d\n",
    health);

  /* Save2file -- Stimulus */
  if(savestim) {
    fpstim = fopen("stim.dat","w");

    if (bin_save) { /* AS 5/00 */
      fwrite(&nstimpts, sizeof(nstimpts), 1, fpstim);    /* long */
      fwrite(&SPER, sizeof(SPER), 1, fpstim);            /* double */
      fwrite(stim, sizeof(stim[0]), nstimpts, fpstim);   /* double(s) */
    }
    else {
      fprintf(fpstim, "%.4f",freq);
      DOFOR(i,nstimpts) fprintf(fpstim, " %e",i*SPER*1e3);
      fprintf(fpstim,"\n");
      fprintf(fpstim, "%.4f",levdbS);
      DOFOR(i,nstimpts) fprintf(fpstim," %e",stim[i]);
      fprintf(fpstim,"\n");
    }

    fclose(fpstim);
  }
```

```
/* Save2file -- BM Filter Outputs */
if(savebm) {
  fpbm = fopen("bm.dat","w");

  if (bin_save) { /* AS 5/00 */
    fwrite(&nstimpts, sizeof(nstimpts), 1, fpbm);    /* long */
    fwrite(&numchs, sizeof(numchs), 1, fpbm);        /* long */
    fwrite(&SPER, sizeof(SPER), 1, fpbm);            /* double */
  }
  else {
    fprintf(fpbm,"%d",numchs);
    DOFOR(i,nstimpts) fprintf(fpbm, " %f",i*SPER*1e3);
    fprintf(fpbm,"\n");
  }
}

/* Save2file -- IHC Outputs */
if(saveihc) {
  fpihc = fopen("ihc.dat","w");

  if (bin_save) { /* AS 5/00 */
    fwrite(&nstimpts, sizeof(nstimpts), 1, fpihc);    /* long */
    fwrite(&numchs, sizeof(numchs), 1, fpihc);        /* long */
    fwrite(&SPER, sizeof(SPER), 1, fpihc);            /* double */
  }
  else {
    fprintf(fpihc,"%d",numchs);
    DOFOR(i,nstimpts) fprintf(fpihc, " %f",i*SPER*1e3);
    fprintf(fpihc,"\n");
  }
}

/* Save2file -- IFR */
if(saveifr) {
  fpifr = fopen("ifr.dat","w");

  if (bin_save) { /* AS 5/00 */
    fwrite(&nstimpts, sizeof(nstimpts), 1, fpifr);    /* long */
    fwrite(&numchs, sizeof(numchs), 1, fpifr);        /* long */
    fwrite(&SPER, sizeof(SPER), 1, fpifr);            /* double */
  }
  else {
    fprintf(fpifr,"%d",numchs);
    DOFOR(i,nstimpts) fprintf(fpifr, " %e",i*SPER*1e3);
    fprintf(fpifr,"\n");
  }
}

/* Channel Loop */
```

```
  DOFOR(ichan,numchs) {
    Cf = anfCfs[ichan];
/*      x=cmaph_f2x(Cf); */
    x = 11.9 * log10(.8 + (Cf/456.));
    printf("chan = %d out of %d  Cf= %f\n",ichan+1,numchs,Cf);

    /* Calculate IFR responses for stimulus */
    gamma4();
    synapse();

    /* Save2file -- BM Filter Outputs */
    if(savebm) {
      if (bin_save) { /* AS 5/00 */
        fwrite(&anfCfs[ichan], sizeof(anfCfs[0]), 1, fpbm);  /* double */
        fwrite(gtf, sizeof(gtf[0]), nstimpts, fpbm);         /* double(s) */
      }
      else {
        fprintf(fpbm,"%f",anfCfs[ichan]);
        DOFOR(i,nstimpts) fprintf(fpbm," %e",gtf[i]);
        fprintf(fpbm,"\n");
      }
    }

    /* Save2file -- IHC Outputs */
    if(saveihc) {
      if (bin_save) { /* AS 5/00 */
        fwrite(&anfCfs[ichan], sizeof(anfCfs[0]), 1, fpihc);  /* double */
        fwrite(ihcL, sizeof(ihcL[0]), nstimpts, fpihc);       /* double(s) */
      }
      else {
        fprintf(fpihc,"%f",anfCfs[ichan]);
        DOFOR(i,nstimpts) fprintf(fpihc," %e",ihcL[i]);
        fprintf(fpihc,"\n");
      }
    }

    /* Save2file -- IFR */
    if(saveifr) {
      if (bin_save) { /* AS 5/00 */
        fwrite(&anfCfs[ichan], sizeof(anfCfs[0]), 1, fpifr);  /* double */
        fwrite(ifr, sizeof(ifr[0]), nstimpts, fpifr);         /* double(s) */
      }
      else {
        fprintf(fpifr,"%f",anfCfs[ichan]);
        DOFOR(i,nstimpts) fprintf(fpifr," %e",ifr[i]);
        fprintf(fpifr,"\n");
      }
    }
```

```
  } /* end ichan loop */

  if(savebm)      fclose(fpbm);
  if(saveihc)      fclose(fpihc);
  if(saveifr)     fclose(fpifr);

  /* free alloc'd arrays   AS 3/00 */
  if (stim) free(stim);
  if (gtf) free(gtf);
  if (ihcL) free(ihcL);
  if (ihc2) free(ihc2);
  if (ifr) free(ifr);

  printf("\n AN filter bank simulation is complete. \n");

}  /* end of main() */

void stimulus(void)
{
/* stimulus amplitudes are in "pascals"
     Conversion: 90dB SPL re 20uPa = .632 Pa */

char waveform_file[40];
  int i;

  printf("Stimulus Waveform Filename: ");
  scanf("%s",waveform_file); printf(" %s\n",waveform_file);
  printf("Time Step Size in input file (secs): ");
  scanf("%lf", &SPER); printf(" %f\n",SPER);
  printf("Input Duration of Simulation (in msec): ");
  scanf("%lf", &stimdur); printf(" %f\n",stimdur);
  printf("  [Zeroes will be added to end of input waveform out to this duration.]\n ");
  /* C convert to seconds */
  stimdur=stimdur*1e-3;

  nstimpts=round(stimdur/SPER);

  /* ***** ALLOCATE MEMORY ***** (AS 3/00) */
  stim = calloc(nstimpts, sizeof(double));
  if (!stim) error ("Memory allocation error in stimulus()");

  DOFOR(i,nstimpts) stim[i] = 0.;     /* zero out buffer */

  printf("\nReady to read in from the waveform file: %s\n  First ten values are:\n",
    waveform_file);

  fpstim = fopen(waveform_file,"r");
  DOFOR(i,nstimpts) {
    /* For single column, ascii, floating point input waveform, use next line */
                fscanf(fpstim,"%lf",&stim[i]);
```

```
                if(i < 10) printf("i = %d  stim[i] = %f\n",i,stim[i]);
    }
  printf("\n");
  fclose(fpstim);
  return;
}  /* End stimulus() */


/* NEW GAMMA4 5/22/98 */
void gamma4(void)
{
  long i,j,idelay;
  double A0,A1,ss0,cc0,ss1,cc1;
  double c, Fc, c1LPihc, c2LPihc, c1LPfb=0, c2LPfb=0;
  double fb = 0., fbl = 0., fbtemp = 0., fbtempl = 0.;
  double wavenow = 0.,wavel = 0.;
  double x = 0., tauLL = 0.;
  double delay = 0., tau0 = 0., tau = 0., Kihc=0, Kfb=0.;
  double asymihc=0., taurange=0, betafb=0;
  double asymfb=0, betaihc=0;
  double Fcfb=0, DC=0.;
  struct complex gtf2[5],gtf2l[5];


/* Initialization */
  for(i = 0; i < 5; i++)
    {
      gtf2[i].x = 0.; gtf2[i].y = 0.; gtf2l[i].x = 0.; gtf2l[i].y = 0.;
    }

  /* parameters for Tau0 vs. CF */
  ss0 = 6.;
  cc0 = 1.1;
  ss1 = 2.2;
  cc1 = 1.1;

  c = 2. / SPER;  /* for Bilinear transformation */
/* IHC LP filter parameters */
  Fc = 4800.; /* Fc is nominal cutoff freq (i.e., -3*n dB down point,
                 n=order) for IHC low-pass filters (Hz) to obtain a
                 3-dB cutoff frequency of 2500 Hz according to Weiss
                 and Rose (1988), and to match Johnson (1980) by
                 eye. */
  c1LPihc = ( c - 2 * PIe * Fc ) / ( c + 2 * PIe * Fc );
  c2LPihc = 2 * PIe * Fc / (2 * PIe * Fc + c );

/* FB LPF parameters */
  Fcfb = 1./(2 * PIe * .002); /* for 2 msec time constant in FB */
  c1LPfb = ( c - 2 * PIe * Fcfb ) / ( c + 2 * PIe * Fcfb );
  c2LPfb = 2 * PIe * Fcfb / (2 * PIe * Fcfb + c );
```

```
/* Find tau0 for this CF */
  /* x = 11.9 * log10(0.80 + Cf / 456.); */ /* position of cf unit;
      from Liberman's map */
  /* Cat filters */
  /* tau0 = ( cc0 * exp( -x / ss0) + cc1 * exp( -x /ss1) ) * 1e-3; */
  /* in sec */

  /* This is the setting for HUMAN filters. Bandwidths are based on
     Glasberg and Moore's (1990) ERB=f(CF) equation.  These values of
     ERB are used at low levels, and then the "high-level" ERB
     corresponding to tau0 is set to be twice as wide. */
  /* tau is (2pi*(1.019*ERB)) */
  /* tau0 in sec */
  /* Gammatone bandwidth is 1/(2*pi*tau) */
  tauLL = 1./(2*PIe*1.019*erbGM(Cf));

/* Set parameters for IHC and FB nonlinearities */
  asymihc = 3; /* asymmetry - this sets positive:negative asymmetry of
                  ihc NL*/
  betaihc = tan(PIe * (-0.5 + 1./(asymihc + 1.))); /* used below to bias NL */
  Kihc = 1225.;
  /* Gain on input of NL- effectively determines threshold & dynamic
     range of IHC - set this using anrhode - saturates at ~60 dB SPL
     (see Dallos) - also influences alpha for rate-level function */

  asymfb = 3; /* asymmetry  - this sets positive:negative asymmetry of ihc NL*/
  betafb = tan(PIe * (-0.5 + 1./(asymfb + 1.)));
  Kfb = 3000.; /* set using anrhode - this determines threshold and
                  range of compression*/
  taurange = 0.5 * tauLL; /*range of tau variation - determines
                            'strength' & influences threshold of
                            compression. Adjust using anrhode and anra
                            (phase) */
  tau0=tauLL-taurange;

  DC = (1.-1./asymfb)/2.; /* - this is asymptotic DC when max of NL is normed to 1 */

  /* i=0 */
  gtf2l[0] = compmult( stim[0], compexp( -2*PIe * Cf * SPER));  /* init */

  for(i = 1; i < nstimpts; i++)   /* Time Loop */
    {
      /* FREQUENCY SHIFT THE ARRAY BUF  */
      gtf2[0] = compmult(stim[i], compexp( -2*PIe * Cf * SPER * i));
      if(health == 1) tau = tau0 + taurange * (DC - fbl)/DC ;
      if(health == 0) tau = tau0;
      if(health == -1) tau = tau0 + taurange;
      if(health ==-2) tau = tau0;
```

```
    for(j = 1; j < 5; j++)  /* IIR Bilinear transformation LPF */
      gtf2[j] = comp2sum(compmult(1./(tau*c+1.),comp2sum(gtf2[j-1],gtf2l[j-1]) ),
                         compmult((tau*c-1.)/(tau*c+1.),gtf2l[j]));

    /* FREQUENCY SHIFT BACK UP   */
    /* Factor of tau put in front of filter 11/26/97
          - normalization by tau0 included for now */
    gtf[i] = tau*tau*tau*tau/(tau0*tau0*tau0*tau0)
      * REAL(compprod(compexp(2*PIe * Cf * SPER * i), gtf2[4]));

    if(health==-2) gtf[i]=gtf[i]*(tau0+taurange)*(tau0+taurange)*(tau0+taurange)*
      (tau0+taurange)/(tau*tau*tau*tau);

    fbtemp = gtf[i]; /* filter output used for feedback */

    /* IHC NL */
    wavenow = (atan(Kihc * gtf[i] + betaihc)
               - atan(betaihc))/(PIe/2. - atan(betaihc));

    /* FB NL */
    fbtemp = (atan(Kfb * fbtemp + betafb)
               - atan(betafb))/(PIe/2. - atan(betafb));

    /* The following LPFs are IIR Bilinear transformation filters */
    ihcL[i] = c1LPihc * ihcL[i-1]
      + c2LPihc * (wavenow + wavel);  /* lp filter the IHC*/

    fb = c1LPfb * fbl
      + c2LPfb * (fbtemp + fbtempl);  /* lp filter the fb tau signal*/

    /* save all loop parameters */
    for(j = 0; j < 5; j++) gtf2l[j] = gtf2[j];
    wavel = wavenow;
    fbl = fb;
    fbtempl = fbtemp;
  }    /* END of TIME LOOP */


/* lowpass filter the IHC voltage  more (these could be merged
   with loop above....) */
/* There's no need to keep entire arrays for the intermediate IHC
   signals, once everything's debugged */

for( i = 1; i < nstimpts; i++)
  ihc2[i] = c1LPihc * ihc2[i-1] + c2LPihc * (ihcL[i] + ihcL[i-1]);

for( i = 1; i < nstimpts; i++)
  ihcL[i] = c1LPihc * ihcL[i-1] + c2LPihc * (ihc2[i] + ihc2[i-1]);

for( i = 1; i < nstimpts; i++)
```

```
    ihc2[i] = c1LPihc * ihc2[i-1] + c2LPihc * (ihcL[i] + ihcL[i-1]);

  for( i = 1; i < nstimpts; i++)
    ihcL[i] = c1LPihc * ihcL[i-1] + c2LPihc * (ihc2[i] + ihc2[i-1]);

  for( i = 1; i < nstimpts; i++)
    ihc2[i] = c1LPihc * ihc2[i-1] + c2LPihc * (ihcL[i] + ihcL[i-1]);

  for( i = 1; i < nstimpts; i++)
    ihcL[i] = c1LPihc * ihcL[i-1] + c2LPihc * (ihc2[i] + ihc2[i-1]);

  /* DELAY THE WAVEFORM (delay gtf and ihcL for display purposes)  */
  /* Note: Latency vs. CF for click responses is available for Cat only (not human) */
  /* Use original fit for Tl (latency vs. CF in msec) from Carney & Yin '88
     and then correct by .75 cycles to go from PEAK delay to ONSET delay */
/*    A0 = 8.13;  */ /* from Carney and Yin '88 */
/*    A1 = 6.49; */
/*    delay = A0 * exp( -x/A1 ) * 1e-3 - 1./Cf;    */
/*    printf("delay=%e\n",delay); */
/*    idelay = delay / SPER; */
/*    for(i = nstimpts; i > idelay; i--) { */
/*      gtf[i] = gtf[i-idelay];  */
/*      ihcL[i] = ihcL[i-idelay];  */
/*    } */
/*    for(i = 1; i < (idelay + 1); i++) { */
/*      gtf[i] = 0.;  */
/*      ihcL[i] = 0.;  */
/*    } */
  /*   printf("No delay\n");  */ /*reminder message, if delay is commented out*/

  return;
}  /* End of gamma4() */


/* NEW SYNAPSE 5/22/99 */
void synapse(void)
{
  long i,j,isp;
  double PIrest,PPI,PImax,PL,PG,CI,CL,CG,VI,VL,Pfactor,Vfactor;
  double c0,s0,c1,s1,dead,rtime,rsptime,rint,prob;
  double g,spont,ftemp,Rsat;
  double p1,p3;
  long option; /* Option for PPI: 1: half-wave rectify, 2: NL Smoother */

  spont = 50.;  /* "spont" rate, before adaptation, refractoriness */
/*    Rsat=165.; */

  Pfactor = .03;  /* ** Controllable ** This might be scaled later to
                     go from inst. rate mode to spikes mode*/
  PIrest = Pfactor/2.5;  /* .012; this will be lower for spikes version */
```

```
  PG = Pfactor;        /* .03 */
  PL = Pfactor * 2.;   /*.06 roughly from W&S 1988 figs */

  Vfactor = 0.0005;   /* **Controllable**  */
  VI = Vfactor;
  VL = 10. * Vfactor;

  CI = spont / PIrest;
  CL = CI * (PIrest + PL)/PL;  /* for stability in steady-state */
  CG = CL * (1. + PL/PG) - CI * PL / PG;  /* so that system is in steady
                                             state at spont */
/*    PImax=PL*PG/(CG*PL*PG/Rsat-PG-PL); */   /* 0.18 newsyn5 */
  PImax=0.6;

  ifr[0] = spont;

  option=2;

  for( i = 1; i < nstimpts ; i ++)
    {

      if (option==1) {
      /* Option 1: Linear equation between ihcL (in range [-1/3,1]) and
         PPI, the half-wave rectify PPI */

        PPI = (PImax-PIrest) * ihcL[i] + PIrest;
        if (PPI<0.0) PPI=0.0;
      }

      if (option==2) {
      /* Option 2: NonLinear relation between ihcL (in range [-1/3,1])
         and PPI, such that PImax and PIrest are achieved and the PPI
         goes to 0 smoothly for negative ihcL */

        p1=log(exp(log(2)*PImax/PIrest)-1);
        p3=p1*PIrest/log(2);

        PPI = p3/p1*log(1+exp(p1*ihcL[i]));
      }

      CI = CI + (SPER/VI)*(-PPI*CI + PL*(CL - CI));
      CL = CL + (SPER/VL)*(-PL*(CL - CI) + PG*(CG - CL));
      ifr[i] = CI * PPI;
    }

  /* Now, ifr[i] contains instantaneous discharge rate vs. time */

  return;
}  /* End of synapse() */
```

```
long round(double value)
{
  if((value-floor(value))>=0.5) return(ceil(value));
  else return(floor(value));
}

double erbGM(double CF)
{
  double erbCf;

  erbCf=24.7*(4.37*Cf/1000+1);

  return(erbCf);
}

double cmaph_f2x(double f)
{
  double x;

  if((f>20677)||(f<20)) error("frequency out of human range, [in cmaph_f2x(f)]");
  x=(1.0/0.06)*log10((f/165.4)+0.88);
  return(x);
}

double cmaph_x2f(double x)
{
  double f;

  if((x>35)||(x<0)) error("BM distance out of human range, [in cmaph_x2f(x)]");
  f=165.4*(pow(10,(0.06*x))-0.88);
  return(f);
}

struct complex compexp(double theta)
     /* this returns a complex number equal to exp(i*theta)  */
{
  struct complex answer;

  answer.x = cos(theta);
  answer.y = sin(theta);
  return answer;
}

struct complex compmult(double scalar, struct complex compnum)
     /* Multiply a complex number by a scalar */
{
  struct complex answer;

  CTREAL(answer,compnum,scalar);
```

```
  return answer;
}

struct complex compprod(struct complex compnum1, struct complex compnum2)
     /* Find the product of 2 complex numbers */
{
  struct complex answer;

  CMULT(answer,compnum1,compnum2);
  return answer;
}

struct complex comp2sum(struct complex summand1, struct complex summand2)
     /* add 2 complex numbers  */
{
  struct complex answer;

  CADD(answer,summand1,summand2);
  return answer;
}

double REAL(struct complex compnum)
{
  return compnum.x;
}

/* error: print an error message and die gracefully */
/* Takes arguments like printf */
/* Copied from Kernighan and Ritchie, p 174 */
void error(char *fmt, ...)
{
  va_list args;
  va_start(args, fmt);
  fprintf(stderr, "error: ");
  vfprintf(stderr, fmt, args);
  fprintf(stderr, "\n");
  va_end(args);
  exit(1);  /* closes all open file */
}
```

Binary file input in Matlab: ANload_bin.m

```
function [bm, time, ANcfs, stim, ifr, ihc] = ANload_bin(base);
% File: ANload_bin.m
% Created by: A. Salomon
% For use with: anmodheinz00.c (by M. G. Heinz)
%
% Loads filterbank responses from the AN model -- BINARY VERSION
%    BM: Basilar membrane
```

```
%   IHC: Inner hair cell
%   IFR: AN instantaneous firing rate
%
% [bm, time, ANcfs[, stim[, ifr[, ihc]]]] = ANload_bin(base);

%%%%%%%%%%%%%%%
% Load and organize all data
%%%%%%%%%%%%%%%
[bm, time, ANcfs] = bin_load_mat( strcat(base, 'bm.dat') );
if (nargout > 3), stim = bin_load_stim ( strcat(base, 'stim.dat') ); end;
if (nargout > 4), ifr = bin_load_mat ( strcat(base, 'ifr.dat') ); end;
if (nargout > 5), ihc = bin_load_mat ( strcat(base, 'ihc.dat') ); end;
```

Appendix B

MATLAB SOURCE (SIGNAL ANALYSIS, ETC.)

B.1 High-level signal analysis code

Main loop: `run_analysis.m`

```
% run_analysis(path to files, delta)

function run_analysis(fpath, delta);

% hard-coded parameters
PER_THRESH = 7.5;
APER_THRESH = 4.0;
PK_THRESH = 3.5;
DIP_THRESH = 2.5;

disp(sprintf('Loading %sbm.dat.', fpath));
[sig_bm, sig_time, sig_ANcfs, sig_stim] = ANload_bin(fpath);

% time-scale dependent defaults
t_ms = (find(sig_time == sig_time(1) + 10.0) - 1) / 10;
p_window = boxcar(round(t_ms*20));     % window for pitch detection
s_step = 2.5;                          % step for pitch summary (ms)

% FIR version of Hilbert transform
  % kaiser window
  A512 = 8 + 512*(2.285 * pi/128);          % pi/128 width transition
  B512 = 0.5842*(A512-21)^0.4 + 0.07886*(A512-21);
  k512 = kaiser(513, B512);

  % transform
  n512 = (0:512) - 256;  n512(512/2+1) = Inf;  % prevent divide by 0
  h512 = k512' .* ((2/pi) * ((sin(pi*(n512)/2).^2) ./ (n512)));
  h512(1:2:513) = 0;  mh = (512/2+1);  % midpoint of hilbert filter

  % simple IIR filter for smoothing
  [B,A] = butter(1, 2000/8000);

  env_bm = zeros(size(sig_bm)); % make space for envelopes
```

```
diffs = [];     d_times = [];
rgns = [];      r_times = [];     r_vconfs = [];

for ch = 1:length(sig_ANcfs),
  disp(sprintf('\n\nChannel %d (CF = %d Hz): \n', ch, sig_ANcfs(ch)));

    % compute envelope (with time correction)
    disp 'Computing envelope.'

    ht = filter(h512, 1, sig_bm(ch,:));
    henv = abs(sig_bm(ch,1:length(sig_bm)-(mh-1)) + j*ht(mh:length(ht)));
    env_bm(ch,1:length(henv)+delta(ch)) = ...
      henv(1-delta(ch):length(henv));
%      filtfilt(B, A, henv(1-delta(ch):length(henv)));  % not filtering

  disp 'Computing pitch information.'
  [rgn,r_time,r_vconf] = channel_pitch(env_bm(ch,:), ...
                           sig_time(1:length(env_bm)), B, A, p_window, 0);
  n_rgns(ch) = length(rgn);
  rgns(ch,1:length(rgn)) = rgn;
  r_times(ch,1:length(rgn)) = r_time;
  r_vconfs(ch,1:length(rgn)) = r_vconf;

  disp 'Computing difference information.'
  diffs(ch,:) = channel_diff(env_bm(ch,:), sig_time, rgn, r_time, ...
                             round(t_ms), 0, 'box');
  d_time = sig_time((0:length(diffs)-1)*round(t_ms)+1);

end;  % for ch


save(strcat(fpath,'env_bm.mat'), 'env_bm', 'B', 'A', 'h512', ...
     'sig_stim', 'sig_time', 'sig_ANcfs');
save(strcat(fpath,'out_ch.mat'), 'n_rgns', 'rgns', 'r_times', ...
     'r_vconfs', 'diffs', 'd_time');

% compute summary information
disp ' '
disp( sprintf('Computing summary measures (step = %.1fms):', s_step) )
[s_per, s_aper, s_pp, ch_class] = ...
    summarize_pitch(n_rgns, rgns, r_times, r_vconfs, sig_ANcfs, ...
      t_ms*1000, length(p_window)/t_ms, s_step);
s_pitch = (t_ms*1000)./s_pp;
s_time = 0:s_step:s_step*(length(s_per)-1);
end;

s_ms_pitch = medsmooth(s_pitch .* (s_per > PER_THRESH), 5);
s_ms_aper = medsmooth(s_aper .* (s_aper > APER_THRESH), 5);
```

```
% compute onset (pos. difference) and offset (neg. difference) components
disp('Computing onset/offset parameters.');
sm_diffs = filtfilt(hamming(5)/sum(hamming(5)), 1, diffs);
s_on = filtfilt(hamming(5)/sum(hamming(5)), 1, mean(sm_diffs .* (sm_diffs > 0)));
s_off = filtfilt(hamming(5)/sum(hamming(5)), 1, mean(sm_diffs .* (sm_diffs < 0)));
s_on_pk = mermel_sh(s_on, PK_THRESH, DIP_THRESH);
s_off_pk = mermel_sh(-s_off, PK_THRESH, DIP_THRESH);

save(strcat(fpath,'output.mat'), ...
     's_time', 's_per', 's_aper', 's_pitch', 'ch_class', ...
     's_ms_pitch', 's_ms_aper', ...
     's_on', 's_off', 's_on_pk', 's_off_pk');
```

Per-channel pitch analysis: channel_pitch.m

```
% [rgn, r_time, r_vconf] = channel_pitch(env_sig, env_time
%                                        [, B, A[, p_window[, s_type]]])
%
%   Given an envelope signal (env_sig) and corresponding time scale in
% ms (env_time), outputs a series of region labels and their associated
% start times.  Note that in periodic regions, one event/region label
% is output per each pitch estimate (approx. one every pitch period).
%
%   rgn, r_time are the region labels with their times; values in rgn are:
%      0 for silence, -10 for aperiodic signal, and
%      >0 for a periodic signal, value is est. pitch period in ms
%
%   env_sig, env_time is the signal with a vector of sample times (in ms)
%   s_type: 0 - linear BM env, 1 - log BM env, 2 - IFR
%
% Optional parameters:
%   B, A - define filter used to clean up AMDF measure
%   p_window - window for periodicity detection (default is 20ms boxcar)

function [rgn, r_time, r_vconf] = channel_pitch(env_sig, env_time, ...
                                        B, A, p_window, s_type);

% internal 'constants'
SIL_THRES_RATIO = -30;        % dB ratio, threshold to mean detection level
P_CONF_THRES = 0.3;           % threshold for periodicity confidence measure
DEFAULT_PP = 5;               % default pitch period (ms) after initial silence

% assume env_time in ms, find #samples in 1ms (assume integral i.e. sr in kHz)
t_ms = (find(env_time == env_time(1) + 10.0) - 1) / 10;
s_rate = t_ms*1000;

SIL = 0;
APER = -10;
min_per = s_rate/500;  max_per = s_rate/55;
```

```
% default parameters
if nargin < 3, [B,A] = butter(1, 2000/(s_rate/2)); end;
if nargin < 5, p_window = boxcar(round(t_ms * 20)); end;
if nargin < 6, s_type = 0; end;

% first section is always called silence, use to train silence threshold
i = 1;   mode = 0;   % mode: 0 = silence, 1 = periodic/aperiodic
rgn(i) = SIL;  r_time(i) = env_time(1);  r_vconf(i) = 0;

switch (s_type),
case 0,  % linear
  sil_thres = 10^(-90/20); %10^(-85/20); %10^(-80/20);
case 1,  % log
  sil_thres = max(env_sig) + SIL_THRES_RATIO;
case 2,  % IFR
  sil_thres = 60;
end;

% prepare for next section
i = i+1;   t = length(p_window)+1;
delta_t = round(DEFAULT_PP*t_ms);             % default delta is short

% loop until t at end
while t < (length(env_sig)-length(p_window)),

  if mode == 0,    % silence mode
    f = find(env_sig(t:length(env_sig)) > sil_thres);
    if length(f),
      t_next = f(1) + t-1;
    else t_next = length(env_sig); end;

    mode = 1;  % don't assume silence next time

  else             % per or aper mode

    % pitch detection
    amdf = AMDF(env_sig, t, p_window);
    [pp, conf] = AMDF_pitch_est(amdf); %, B, A);

    if conf > P_CONF_THRES & pp >= min_per & pp <= max_per,
      rgn(i) = pp/t_ms;
    else
      rgn(i) = APER;
    end;
    r_time(i) = env_time(t);  r_vconf(i) = conf;  i = i+1;

    delta_t = round( (conf^2)*pp + (1-conf^2)*delta_t );
    t_next = t + delta_t;
  end;
```

```
  % prepare for next iteration
  t = t_next;

  % silence detection
  if max(env_sig(t:min(t+delta_t,length(env_time)))) < sil_thres,
    rgn(i) = SIL;  r_time(i) = env_time(t);  r_vconf(i) = 0;
    i = i+1;  mode = 0;
  end;
end;

disp(sprintf('%d events total.', i-1));
```

Per-channel difference analysis: `channel_diff.m`

```
% [diffs, e_steps] = channel_diff(env_sig, env_time, rgn, r_time, step
%                                  [, diff_type [, sil_e_step, fric_e_step]]);
%
% Optional parameters:
%   diff_type - type of differencing operator - e.g. 'dgauss', 'box', 'hamming'
%   sil_e_step, fric_e_step - energy windowing difference lengths for
%     silence (stop detection) and aperiodic (fricative detection) modes

function [diffs, e_steps] = channel_diff(env_sig, env_time, rgn, r_time, ...
                              step, diff_type, sil_e_step, fric_e_step);

% region type values
SIL = 0;  APER = -10;

% assume env_time in ms, find #samples in 1ms (assume integral i.e. sr in kHz)
t_ms = (find(env_time == env_time(1) + 10.0) - 1) / 10;
len = length(env_sig);

% default parameters
if nargin < 5, step = t_ms; end;
if nargin < 7, diff_type = 'box'; end;
if nargin < 8, sil_e_step = round(5*t_ms); fric_e_step = round(30*t_ms); end;

diffs = zeros(ceil(size(env_time)/step));
e_steps = zeros(ceil(size(env_time)/step));

% add extra region at end, if needed
if r_time(length(rgn)) < max(env_time),
  rgn = [rgn 0];
  r_time = [r_time max(env_time)];
end;

e_step = sil_e_step;         % default e_step is short
[e_wl,e_wr] = diff_window(e_step, diff_type);
```

```
% loop over all regions, pick diff sizes
for i = 1:length(rgn)-1,

  t = find(env_time == r_time(i));
  t_next = find(env_time == r_time(i+1));

  if rgn(i) == APER,          target_e_step = fric_e_step;
  elseif rgn(i) == SIL,       target_e_step = sil_e_step;
  else  target_e_step = round(2*rgn(i)*t_ms);  % 2x pitch period
  end;
%  disp(sprintf('%d,%.2f: target e_step is %d', i, r_time(i), target_e_step))

  t = max(t,e_step+1) + step;

  % energy difference detection
  for n = floor(t / step):floor(t_next / step),
    % step slew control
    if (e_step ~= target_e_step),
      if (e_step < target_e_step),
        e_step = min(e_step+step/2, target_e_step);
      elseif (e_step > target_e_step),
        e_step = max(e_step-step/2, target_e_step);
      end;
    end;

    % are we done?
    loc = (n-1)*step+1;
    if e_step + loc > len, break; end;

    e_steps(n) = e_step;
  end;
end;


% Now that we have times, run diffs (performance improvement, since
% diffs are computed over larger regions)

% split into 'regions'
rgn_locs = find(diff(e_steps)) + 1;
r = 1;

while r < length(rgn_locs),
  % compute window
  n = rgn_locs(r);
  [e_wl,e_wr] = diff_window(e_steps(n), diff_type);

  % filter
  next_n = min(rgn_locs(r+1), length(env_sig)-length(e_wl));
```

```
  ldiff_locs = -(length(e_wl)-1) + (n-1)*step+1 : (next_n-1 -1)*step+1;
  rdiff_locs = (n-1)*step+1 : (next_n-1 -1)*step+1 + length(e_wr)-1;

  left = filter(e_wl, 1, env_sig(ldiff_locs));
  right = filter(e_wl, 1, env_sig(rdiff_locs));

  the_diffs = 20*log10(max(1e-6,right)) - 20*log10(max(1e-6,left));

  diffs(n:next_n-1) = the_diffs(length(e_wl):step:length(the_diffs));

  % next 'rgn'
  r = r+1;
end;
```

Pitch summary: summarize_pitch.m

```
% [s_per, s_aper, s_pp, ch_class] =
%   summarize_pitch(n_rgns, rgns, r_times, r_vconfs, sig_ANcfs,
%       [s_rate, [pw_len, [step]]]);
function [s_per, s_aper, s_pp, ch_class] = ...
   summarize_pitch(n_rgns, rgns, r_times, r_vconfs, sig_ANcfs, ...
   s_rate, pw_len, step, start, end_t);

% hard-coded parameters
P_CONF_THRESH = 0.3;

% region type values
SIL = 0;  APER = -10;


% defaults
if nargin < 6, s_rate = 16000; end;
if nargin < 7, pw_len = 20; end;
if nargin < 8, step = 1; end;
if nargin < 9, start = 0; end;
if nargin < 10, end_t = Inf; end;

% compute size of output
nchs = length(n_rgns);
len = ceil( ( min( end_t,max(r_times(:)) )-start )/step );
t_ms = s_rate/1000;   % for use in region time translation (back to samples)
     % TODO: just do all computations in samples

% default output values
s_per = zeros(1,len);
s_aper = zeros(1,len);
s_pp = Inf .* ones(1,len);
ch_class = zeros(nchs,len);
```

```
% set unused region times to end of signal
for n = 1:nchs,
  r_times(n,n_rgns(n)+1:length(r_times)) = len;
end;

% bracket beginning and end of interest in each estimate
start_times = r_times - max(0,rgns);
end_times = r_times + pw_len;

% loop over time
for n = 0:len-1,
  % find relevant estimates (TODO: don't use low conf est?), get pitch est
  est_locs = find(start_times(:) <= start+n*step & ...
                  end_times(:) > start+n*step & ...
  rgns(:) > 0); % & r_vconfs(:) > 0.1);
  [pp, ph_conf, which] = pitch_hist( ...     % including pp/2 est @ conf/2
    [rgns(est_locs)*t_ms; fix(rgns(est_locs)*t_ms/2)], ...
    [r_vconfs(est_locs); r_vconfs(est_locs)/3]);
  which = rem(which-1, length(est_locs))+1;
%  disp(sprintf('est_locs {%d}, which {%d}', length(est_locs), length(which)))

  % store pitch estimate
  if ph_conf, s_pp(n+1) = pp;
  else        s_pp(n+1) = Inf; end;    % division -> pitch = 0

  % store channel info
  % - find silent channels
  sil_locs = find(r_times(:,1:length(r_times)-1) <= start+n*step & ...
  start+n*step < r_times(:,2:length(r_times)) & ...
                  rgns(:,1:length(rgns)-1) == SIL);
  sil_chs = unique( rem(sil_locs-1,nchs)+1 );
  ch_class(sil_chs,n+1) = -1;

  % - find periodic channels
  est_ch = rem(est_locs(which)-1,nchs)+1;
  per_chs = unique(est_ch);
  for i = 1:length(per_chs),
    ch_ests = est_locs(which( find(est_ch == per_chs(i)) ));
    ch_class(per_chs(i),n+1) = median(r_vconfs(ch_ests));
  end;

  % compute periodicity confidence (sum of conf for periodic channels)
  s_per(n+1) = sum(ch_class(per_chs,n+1));

  % - find aperiodic channels
  aper_chs = find(ch_class(:,n+1) >= 0 & ch_class(:,n+1) < P_CONF_THRESH & ...
                  sig_ANcfs > (s_rate / s_pp(n+1)));
  if ~isempty(est_locs),
    est_ch = rem(est_locs-1,nchs)+1;
    for i = 1:length(aper_chs),
```

```
      ch_ests = est_locs( find(est_ch == aper_chs(i)) );
      if isempty(ch_ests), ch_class(aper_chs(i),n+1) = -1;
      else,           ch_class(aper_chs(i),n+1) = - mean(r_vconfs(ch_ests));
      end;
    end;

    % compute aperiodicity confidence (sum of conf for aperiodic channels)
    s_aper(n+1) = sum(ch_class(aper_chs,n+1) + 1);
  end;

  if ~rem(n*step,250), disp(sprintf('%d,', start+n*step)); end;
end;
```

B.2 Event detection

Event detection: `LM_output.m` (with hooks for training)

```
% [ev_times, ev_values, ev_labels] = ...
%         LM_output(f_name, s_time, s_per, s_aper, s_pitch, ...
%     d_time, s_on, s_off, s_on_pk, s_off_pk);
%

function [ev_times, ev_values, ev_labels] = LM_output(f_name, postfix, ...
   PON_TIME_THR_pre, PON_TIME_THR_pos, POFF_TIME_THR, AP_TIME_THR, ...
   PER_THRESH_HIGH, PER_THRESH, APER_THRESH_HIGH, APER_THRESH, ...
   ON_PK_THRESH, ON_DIP_THRESH, OFF_PK_THRESH, OFF_DIP_THRESH);

load(strcat(f_name,'.out_ch.mat'));
load(strcat(f_name,'.output.mat'));

f_name = strcat(f_name, '.LM', postfix);

if nargin < 3, PON_TIME_THR_pre = 10; end; % max time diff from Pon to onset for v+ events
if nargin < 4, PON_TIME_THR_pos = 10; end; %    _pre -> preceding, _pos -> following
if nargin < 5, POFF_TIME_THR    = 30; end; % max time diff from Poff to off for v- events
if nargin < 6, AP_TIME_THR      = 20; end; % max time diff from AP to on/f for c+/- events
if nargin < 7,  PER_THRESH_HIGH  = 7.5; end;
if nargin < 8,  PER_THRESH       = 7.5; end;
if nargin < 9,  APER_THRESH_HIGH = 8.0; end;
if nargin < 10, APER_THRESH      = 5.0; end;
if nargin < 11, ON_PK_THRESH     = 3.5; end;
if nargin < 12, ON_DIP_THRESH    = 2.5; end;
if nargin < 13, OFF_PK_THRESH    = 3.5; end;
if nargin < 14, OFF_DIP_THRESH   = 2.5; end;

% compute peaks
s_on_pk = mermel_sh(s_on, ON_PK_THRESH, ON_DIP_THRESH);
s_off_pk = mermel_sh(-s_off, OFF_PK_THRESH, OFF_DIP_THRESH);
```

```
sl = length(s_time);
dl = length(d_time);

% not settable
SON_TIME_THR = 20;     % region around on/off to call an event s+/-
P_MIN_TIME = 10;       % shortest Pon-Poff time
AP_MIN_TIME = 10;      % shortest APon-APoff time
SON_THR = 0.8;         % proportion of slices labelled per. to count as son.

% median smoothing / temporal thresholds
s_ms_pitch = medsmooth(s_pitch .* (s_per > PER_THRESH), 5);
s_ms_aper = medsmooth(s_aper .* (s_aper > APER_THRESH), 5);

% open output file
fid = fopen(f_name, 'w');

% find region boundaries
Pon_loc = (find( s_ms_pitch(1:sl-1) == 0  &  s_ms_pitch(2:sl) > 0 ));
Poff_loc = (find( s_ms_pitch(1:sl-1) > 0  &  s_ms_pitch(2:sl) == 0 ) +1);
APon_loc = (find( s_ms_aper(1:sl-1) == 0  &  s_ms_aper(2:sl) > 0 ));
APoff_loc = (find( s_ms_aper(1:sl-1) > 0  &  s_ms_aper(2:sl) == 0 ) +1);

% initialize landmark lists
n_evs = 0;
ev_times = [];   ev_values = [];  ev_labels = {};

% prune P events if pitch within region doesn't agree with most of utt,
% or max Pconf too low
Pprune = zeros(max(length(Pon_loc),length(Poff_loc)),2);
exp_pitch = median(nonzeros(s_ms_pitch));
for n = 1:length(Pon_loc),
  n_off = min(find(Poff_loc > Pon_loc(n)));   % should be = n?  ASSUME YES

  % pruning rule:
  rgn_pitch = median( s_ms_pitch(Pon_loc(n):Poff_loc(n_off)) );
  if rgn_pitch > 2*exp_pitch  |  rgn_pitch < .5*exp_pitch  | ...
       (Poff_loc(n_off) - Pon_loc(n)) < P_MIN_TIME | ...
       max( s_per(Pon_loc(n):Poff_loc(n_off)) ) < PER_THRESH_HIGH,
    s_ms_pitch(Pon_loc(n):Poff_loc(n_off)) = 0;  % explicit prune!!
    Pprune(n,1) = 1; Pprune(n_off,2) = 1;
  end;
end; % for

Pon_loc = Pon_loc( find(~Pprune(:,1)) );
Poff_loc = Poff_loc( find(~Pprune(:,2)) );

% loop over P/AP events -- associate w/ on/off if they exist
%  -> otherwise, insert w/ strength value of 0
%    NOTE: per evts are 'v', aper are 'c'
```

```
% - Pon
for n = 1:length(Pon_loc),
  pk_offs = s_time(Pon_loc(n)) - d_time(s_on_pk);
  ind_pks = find(pk_offs > -PON_TIME_THR_pre  &  pk_offs < PON_TIME_THR_pos);
  if ~isempty(ind_pks),
    l = max(ind_pks);
    ev_t = d_time(s_on_pk(l));  ev_v = s_on(s_on_pk(l));
    s_on_pk = s_on_pk([1:l-1 l+1:length(s_on_pk)]);
  else,
    ev_t = s_time(Pon_loc(n));  ev_v = 0;
  end;

  n_evs = n_evs+1;
  ev_times(n_evs) = ev_t;  ev_values(n_evs) = ev_v;
  ev_labels{n_evs} = '+v';
end;

% - Poff
for n = 1:length(Poff_loc),
  [d,l] = min(abs( s_time(Poff_loc(n)) - d_time(s_off_pk) ));
  if d <= POFF_TIME_THR,
    ev_t = d_time(s_off_pk(l));  ev_v = -s_off(s_off_pk(l));
    s_off_pk = s_off_pk([1:l-1 l+1:length(s_off_pk)]);
  else,
    ev_t = s_time(Poff_loc(n));  ev_v = 0;
  end;

  n_evs = n_evs+1;
  ev_times(n_evs) = ev_t;  ev_values(n_evs) = ev_v;
  ev_labels{n_evs} = '-v';
end;

% now that P on/off events have been removed,
% prune AP events unless
%    a) there is a correp on or off event (in s_{on|off}_pk)
%    b) max in region > APER_THRESH_HIGH
%  OR always prune if
%    c) region completely periodic (surr by Pon/off)
%    d) region shorter than 10ms (i.e. 4 frames @ 2.5ms/sample)

APprune = zeros(max(length(APon_loc),length(APoff_loc)),2);
for n = 1:length(APon_loc),
  n_off = min(find(APoff_loc > APon_loc(n)));   % should be = n?  ASSUME YES

  % find nearest on/off events
  [dn,ln] = min(abs( s_time(APon_loc(n)) - d_time(s_on_pk) ));
  [df,lf] = min(abs( s_time(APoff_loc(n_off)) - d_time(s_off_pk) ));

  % pruning rules:
```

```
    % - definitely prune if rgn completely periodic (>80%..)
    % - also prune region if both dn,df above thresh AND max in region too low
  if mean(s_ms_pitch(APon_loc(n):APoff_loc(n_off)) > 0) > SON_THR | ...
     (APoff_loc(n_off) - APon_loc(n)) < AP_MIN_TIME | ...
     ( dn > AP_TIME_THR &  df > AP_TIME_THR & ...
       max( s_ms_aper(APon_loc(n):APoff_loc(n_off)) ) < APER_THRESH_HIGH ),
    APprune(n,1) = 1; APprune(n_off,2) = 1;
    s_ms_aper(APon_loc(n):APoff_loc(n_off)) = 0;  % explicit prune!!
  end;
end; % for

APon_loc = APon_loc( find(~APprune(:,1)) );
APoff_loc = APoff_loc( find(~APprune(:,2)) );

% - APon
for n = 1:length(APon_loc),
  [d,l] = min(abs( s_time(APon_loc(n)) - d_time(s_on_pk) ));
  if d <= AP_TIME_THR,
    ev_t = d_time(s_on_pk(l));  ev_v = s_on(s_on_pk(l));
    s_on_pk = s_on_pk([1:l-1 l+1:length(s_on_pk)]);
  else,
    ev_t = s_time(APon_loc(n));  ev_v = 0;
  end;

  n_evs = n_evs+1;
  ev_times(n_evs) = ev_t;  ev_values(n_evs) = ev_v;
  ev_labels{n_evs} = '+c';
end;

% - APoff
for n = 1:length(APoff_loc),
  [d,l] = min(abs( s_time(APoff_loc(n)) - d_time(s_off_pk) ));
  if d <= AP_TIME_THR,
    ev_t = d_time(s_off_pk(l));  ev_v = -s_off(s_off_pk(l));
    s_off_pk = s_off_pk([1:l-1 l+1:length(s_off_pk)]);
  else,
    ev_t = s_time(APoff_loc(n));  ev_v = 0;
  end;

  n_evs = n_evs+1;
  ev_times(n_evs) = ev_t;  ev_values(n_evs) = ev_v;
  ev_labels{n_evs} = '-c';
end;


% loop over remaining on/off events
[event_locs, ind] = sort( d_time([s_on_pk s_off_pk]) );
event_vals = [s_on(s_on_pk) s_off(s_off_pk)];
event_vals = event_vals(ind);
```

```
for n = 1:length(event_locs),
  ev_class = 'c';
  s_locs = find(abs(s_time-event_locs(n)) < SON_TIME_THR/2);

  % polarity: onset or offset
  if event_vals(n) < 0,   ev_pol = '-';
    if mean(s_ms_pitch(s_locs) > 0) > SON_THR, ev_class = 's';  end;

  else                    ev_pol = '+';
    if mean(s_ms_pitch(s_locs) > 0) > SON_THR, ev_class = 's';  end;
  end;

  n_evs = n_evs+1;
  ev_times(n_evs) = event_locs(n);
  ev_values(n_evs) = abs(event_vals(n));
  ev_labels{n_evs} = strcat(ev_pol, ev_class);
  end;

[ev_times, ind] = sort(ev_times);
ev_values = ev_values(ind);
ev_labels = ev_labels(ind);

for n = 1:n_evs,
  fprintf(fid, '%.2f %.2f %s\n', ...
          ev_times(n), ev_values(n), ev_labels{n});
end;

fclose(fid);
```

B.3 Subroutines used in signal analysis

AMDF function: AMDF.m

```
% amdf = AMDF(s,t,w);
%
% Computes average magnitude difference function (AMDF) over signal s, or
% a set of signals (one FB output per row), starting at time t (in samples)
% with specified window w, as per de Cheveigne' (1998).
%
% w defaults to a rectangular window of length 20ms (assuming 16kHz samp rate).

function amdf = AMDF(s, t, w);

if nargin < 3,
  w = boxcar(20*16)/(20*16);
end;

for i = 1:size(s,1),
```

```
  for tau = 1:length(w),
    amdf(i,tau) = sum( w' .* ...
                abs(s(i,t:t+length(w)-1) - s(i,t-tau:t+length(w)-1-tau)) );
  end;
end;
```

Pitch histogram analysis: AMDF_pitch_est.m

```
% [pp, conf] = AMDF_pitch_est(amdf[, fa, fb]);
%
% Find pitch period peak in an AMDF measure.
%
%   amdf is the amdf function output.
%   fb, fa specify an optional filter to be used to smooth the amdf function.
%
% Outputs:
%   pp is the location of the pitch period peak, in samples
%   conf is a confidence measure based on depth of dip

function [pp, conf] = AMDF_pitch_est(amdf, fa, fb);

% filter with supplied filter
if nargin > 1,
  if nargin < 3, fa = 1; end;
  amdf = filtfilt(fb, fa, amdf);
end;

% convex hull
hull = convex_hull(amdf);

% find max dip
[dip,dip_loc] = max(hull - amdf);

% translate into pp, conf -- find max. confidence
conf_hull = (hull - amdf) ./ hull;
[conf,pp] = max(conf_hull);
```

Difference window generation: diff_window.m

```
% [w_left, w_right] = diff_window(step[, f])
%
%   Compute energy difference window for a difference length equal to step.
%
%   Basically, an adaptive filter..  f can be used to specify the
% filter type: 'dgauss' for derivative of a gaussian, 'box' for a pair
% of box functions, 'hamming' for a pair of hamming windows.

function [w_l, w_r] = energy_diff(step, f);
```

```
% defaults
if nargin < 2,  f = 'dgauss';  end;

% compute filter (difference function)
if strcmp(f, 'dgauss'),
  nl = -step+0.5:-0.5;  nr = 0.5:step-0.5;
  w_l = -1/sqrt(2*pi) * 2*nl/(step/4.1343739) .* exp(-(nl.^2)/(2*(step/2)^2));
  w_r = 1/sqrt(2*pi) * 2*nr/(step/4.1343739) .* exp(-(nr.^2)/(2*(step/2)^2));
elseif strcmp(f, 'box'),
  w_l = boxcar(step)';
  w_r = boxcar(step)';
elseif strcmp(f, 'hamming'),
  w_l = hamming(step)';
  w_r = hamming(step)';
else
%  error(sprintf('Bad filter specification ''%s''.', f));
  w_l = eval(sprintf('%s(%d)', f, step));  w_l = w_l(:)';
  w_r = eval(sprintf('%s(%d)', f, step));  w_r = w_r(:)';
end;

if nargout == 1,  w_l = [-w_l w_r];  end;
```

Median smoothing: `medsmooth.m`

```
function [y] = medsmooth(x, ws);
% MEDSMOOTH Median smoothing
%    [y] = medsmooth(x, ws);

back = floor((ws-1)/2);
forw = ceil((ws-1)/2);

for i = 1:length(x),
  y(i) = median(x(max(1,i-back):min(length(x),i+forw)));
end;
```

Convex hull: `convex_hull.m`

```
function hull = convex_hull(sig);

% convex hull
len = length(sig);

hull(1) = sig(1);
for n = 2:len,
  hull(n) = max(hull(n-1),sig(n));
end;
return;
```

Convex hull peak-picking: `mermel.m`

```
function [pk_locs] = mermel(signal, pk_thresh, dip_thresh);

[max_val,max_loc] = max(signal);

% compute convex hull: foward
i = 1;  h = signal(1);
while(i < max_loc),
  pts = find(signal(i+1:max_loc) > h);
  if (isempty(pts))
    np = max_loc;
  else
    np = i+pts(1);
  end;
  hull(i:np-1) = h;

  i = np; h = signal(np);
end;

% compute convex hull: backward
i = length(signal);  h = signal(i);
while(i > max_loc),
  pts = find(signal(i-1:-1:max_loc) > h);
  if (isempty(pts))
    np = max_loc;
  else
    np = i-pts(1);
  end;
  hull(i:-1:np+1) = h;

  i = np; h = signal(np);
end;

hull(max_loc) = max_val;


% apply thresholds, recurse
[dip,dip_loc] = max(hull - signal);

if (max_val > pk_thresh),
  if (dip > dip_thresh),
    pk_locs = [mermel(signal(1:dip_loc), pk_thresh, dip_thresh) ...
       mermel(signal(dip_loc:length(signal)), pk_thresh, dip_thresh)+dip_loc-1];
  else
    pk_locs = max_loc;
  end;
else
  pk_locs = [];
end;
```

B.4 Pitch scoring

Pitch comparison with *get_f0* output: `compare_f0.m`

```
function [correct, p_rms_err, count, uv_correct, uv_count, ...
       p_rms_err_x2, x2_ref, x2_det, vc_count, m_pitch] = compare_f0(basename);

load(strcat(basename,'.output.mat'));

fid = fopen(strcat(basename,'.wav.f0'));
f0 = fscanf(fid, '%f');
fclose(fid);

count = length(f0);  v_count = nnz(f0>0);  uv_count = nnz(f0==0);
p_offset = find(s_time == 0.01*1000);
p_time = s_time(p_offset:4:p_offset+(count-1)*4);
f0_est = s_ms_pitch(p_offset:4:p_offset+(count-1)*4);
f0_est = f0_est(:);

v_correct = nnz(f0 > 0 & f0_est > 0) / v_count;
uv_correct = nnz(f0 == 0 & f0_est == 0) / uv_count;
correct = ( nnz(f0 > 0 & f0_est > 0) + nnz(f0 == 0 & f0_est == 0) ) / count;

v_idx = find(f0 > 0 & f0_est > 0);
p_rms_err = sqrt(mean((f0(v_idx) - f0_est(v_idx)).^2));
p_rms_err_x2 = sqrt(mean(min( (f0(v_idx) - f0_est(v_idx)).^2, ...
  min((f0(v_idx) - 2*f0_est(v_idx)).^2, (2*f0(v_idx) - f0_est(v_idx)).^2) )));
x2_ref = nnz( (f0(v_idx)-f0_est(v_idx)).^2 > (2*f0(v_idx)-f0_est(v_idx)).^2 );
x2_det = nnz( (f0(v_idx)-f0_est(v_idx)).^2 > (f0(v_idx)-2*f0_est(v_idx)).^2 );

vc_count = length(v_idx);
m_pitch = median(f0(v_idx));

if nargout < 1,
  disp( sprintf('Correct:  %5.2f%%', correct*100) );
  disp( sprintf('(V:  %5.2f%%\tUV: %5.2f%%)', [v_correct uv_correct].*100) );
  disp( sprintf('Pitch RMS error:  %5.2fHz (modified: %5.2fHz)', ...
p_rms_err, p_rms_err_x2) );

  plot(p_time, f0, p_time, f0_est)
end;
```

Appendix C

TOOLS FOR POSITING EVENTS AND SCORING

Project declaration: `Makefile`

```
CXXFLAGS = -g -Wall -ansi

TARGETS = phn2lm compare

build: ${TARGETS}


phn2lm: phn2lm.o
g++ ${CXXFLAGS} -o $@ $^

phn2lm.o: phn2lm.cc context/TIMIT.h compare.h


compare: compare.o align.o
g++ ${CXXFLAGS} -o $@ $^

compare.o: compare.cc compare.h
align.o: align.cc compare.h


clean:
-rm ${TARGETS}
-rm *.o
```

Scoring script: `score.sh`

```
#!/bin/bash

if test "$#" -lt 1; then
    echo Usage: score.sh <base> ...,
    echo    where <base>.phn and <base>.LM must exist for each input
fi

while test $# -ge 1; do
    if test ! -f $1.phn  -a  ! -f $1.PLM; then
      echo "Can't find $1.phn or $1.PLM (aborting)"; exit -1; fi
```

```
    if test ! -f $1.LM; then echo "Can't find $1.LM (aborting)"; exit -1; fi

    if test ! -f $1.PLM; then
        phn2lm $1.phn $1.PLM
    fi

    echo $1
    echo ------------------------------
    compare $1.PLM $1.LM
    echo ------------------------------
    echo

    shift
done
```

Definitions: `compare.h` (includes penalty table class for alignment algorithm)

```
#include <iostream>
#include <string>
#include <vector>


//
// high-level constants -- parameters to algorithm

#define MAX_TIME_DIFF   50.0 /*ms*/


//
// event_type enum and i/o operations

enum event_type { ev_INV = -1,
                  ev_Von, ev_Voff, ev_Con, ev_Coff, ev_Son, ev_Soff,
                  NUM_EV_TYPES };

inline bool polarity( event_type ev )
{
  return ( ev == ev_Von || ev == ev_Con || ev == ev_Son );
}

inline bool isvalid ( event_type ev )
{
  return ( ev > ev_INV && ev < NUM_EV_TYPES );
}

inline ostream& operator << (ostream& os, const event_type ev)
{
  switch (ev) {
  case ev_Von:  os << "+v"; break;
```

```
  case ev_Voff: os << "-v"; break;
  case ev_Con:  os << "+c"; break;
  case ev_Coff: os << "-c"; break;
  case ev_Son:  os << "+s"; break;
  case ev_Soff: os << "-s"; break;
  default:      os << "??";
  }
  return os;
}

inline istream& operator >> (istream& is, event_type& ev)
{
  char type, pol;

  is >> ws >> pol >> type;
  if ( (pol == '+' || pol == '-') )
    switch (type) {
    case 'v': ev = (pol == '+') ? ev_Von : ev_Voff; break;
    case 'c': ev = (pol == '+') ? ev_Con : ev_Coff; break;
    case 's': ev = (pol == '+') ? ev_Son : ev_Soff; break;
    default:  ev = ev_INV;
    }
  else ev = ev_INV;

  if ( (ev == ev_INV) && !(pol == '?' && type == '?') )
    is.putback(type).putback(pol);

  return is;
}


//
// LM_label: recognized landmark structure

struct LM_label {
  LM_label() : ev(ev_INV), time(-1), strength(0.0) {}
  bool isvalid() { return ::isvalid(ev); }

  event_type ev;            // event type and polarity
  float time, strength;     // time, strength value
};

inline istream& operator >> (istream& is, LM_label& lm)
{
  is >> lm.time >> lm.strength >> ws >> lm.ev;
  return is;
}

inline ostream& operator << (ostream& os, const LM_label& lm)
{
```

```
  os << lm.time << '\t' << lm.strength << '\t' << lm.ev;
  return os;
}


//
// LM_ref_label: reference landmark structure

struct LM_ref_label {
  enum { MAX_EV = 2 };

  LM_ref_label() : time(-1) { ev[0] = ev_INV; }
  bool isvalid() { return ::isvalid(ev[0]); }

  event_type ev[MAX_EV];       // event type(s) and polarity(ies)
  bool req[MAX_EV];            // whether event is 'required' (= cost for DEL)
  float time;                  // time, strength value
  string comment;              // comment field, usu adj segment labels
};

inline istream& operator >> (istream& is, LM_ref_label& lm)
{
  int i;

  is >> lm.time >> lm.comment;

  for (i = 0; i < LM_ref_label::MAX_EV; i++) {
    is >> lm.ev[i];
    lm.req[i] = (is.peek() != '?');  if (!lm.req[i]) is.get();
    if (is.peek() == ',') is.get(); else break;
  }

  while (++i < LM_ref_label::MAX_EV) { lm.ev[i] = ev_INV; lm.req[i] = false; }

  return is;
}

inline ostream& operator << (ostream& os, const LM_ref_label& lm)
{
  os << lm.time << '\t' << lm.comment << '\t' << lm.ev[0];
  if (lm.ev[1] != ev_INV) os << ',' << lm.ev[1];
  return os;
}


//
// aggregated objects

typedef vector<LM_label> LM_label_seq;
typedef vector<LM_ref_label> LM_ref_label_seq;
```

```
//
// prototypes


//
// penalty table class

class penalty_table {
  enum {
    MAX_penalty = 1000000,
    SUB_penalty = 50 /*100*/,
    INS_penalty = 50 /*75*/,
    DEL_penalty = 50 /*75*/
  };

  enum p_step { INVALID = -1, OK, OK2, SUB, DEL, INS };

  struct node {
    enum { MAX_EV = LM_ref_label::MAX_EV };

    node() : p_step(INVALID), cost(MAX_penalty) {}

    p_step p_step;
    int cost;
    bool avail[MAX_EV];

    int navail() {
      int n=0;
      for (int i=0; i < MAX_EV; i++) n += (avail[i]?1:0);
      return n;
    }
    int nreq(LM_ref_label& lm) {
      int n=0;
      for (int i=0; i < MAX_EV; i++) n += ((avail[i]&lm.req[i])?1:0);
      return n;
    }

    void fill(int _cost, p_step _p_step, LM_ref_label& _lbl, int ev_use = -1) {
      cost = _cost;   p_step = _p_step;
      for (int i = 0; i < MAX_EV; i++)
        avail[i] = (_lbl.ev[i] != ev_INV) & (i != ev_use);
    }
    void fill(int _cost, p_step _p_step, node& n) {
      cost = _cost;   p_step = _p_step;
      for (int i = 0; i < MAX_EV; i++) avail[i] = n.avail[i];
    }
  };
```

```
public:
  penalty_table() : nbacktr(0) {}

  void create_table(LM_ref_label_seq&, LM_label_seq&);
  int backtrace(LM_ref_label_seq&, LM_label_seq&);

  ostream& output(ostream&, LM_ref_label_seq&, LM_label_seq&);
  ostream& output_results(ostream&);

private:
  enum { ML = 250 };
  node the_table[ML][ML];
  int n1, n2;   // size of ref, recog label strings

  p_step backtr[2*ML];
  int nbacktr, nerr, nsub, nins, ndel, nokins, nokdel;
};
```

TIMIT-format label access: `TIMIT.h`

```
// TIMIT.h: Definitions for reading/writing TIMIT-format labels

#ifndef __TIMIT_H
#define __TIMIT_H

#include <string>
#include <iostream>


const int TIMIT_sample_rate = 16000;

struct TIMIT_label
{
  int start;
  int end;
  string phon;

  float start_time() { return float(start)/TIMIT_sample_rate; }
  float end_time() { return float(end)/TIMIT_sample_rate; }
};

inline istream& operator >> (istream& is, TIMIT_label& label)
{
  is >> label.start >> label.end >> label.phon;
  return is;
}

inline ostream& operator << (ostream& os, TIMIT_label& label)
```

```
{
  os << label.start << ' ' << label.end << ' ' << label.phon << endl;
  return os;
}


#endif // __TIMIT_H
```

C.1 *Positing landmarks (*phn2lm *program)*

Main program: phn2lm.cc

```
//
// phn2lm.cc: convert TIMIT .phn labels to expected landmark sequence
//

#include <fstream>
#include <map>
#include <string>

#include "TIMIT.h"
#include "compare.h"           // get event_type declaration

inline void write_LM(ostream& os, TIMIT_label &prev, TIMIT_label &curr,
                     char *LMtype, char *comment = "", float t = -22)
{
  if (t < 0) t = curr.start_time();

  os << t*1000 << '\t'
     << prev.phon << ';' << curr.phon << '\t'
     << LMtype << endl;
}

enum Manner { INVALID, Closure, StopBurst, Fricative, Sonorant, Vocalic };

class SegmentClass {
public:
  Manner manner()        { return m_manner; }
  bool is_sonorant()    { return m_manner == Sonorant || m_manner == Vocalic; }
  bool is_obstruent()   { return m_manner == Fricative
                            || m_manner == StopBurst; }

  bool voiced()               { return m_voiced; }
  bool strident()        { return m_strident; }
  bool is_canonical()        { return m_ctype.empty(); }
  string& type_label()        { return m_ctype; }
  string& seg_label()        { return m_label; }
```

```
  SegmentClass() : m_manner(INVALID) {}
  SegmentClass(Manner manner, bool voiced, bool strident, string ctype)
    : m_manner(manner), m_voiced(voiced), m_strident(strident), m_ctype(ctype)
    {}

  SegmentClass& operator = (const SegmentClass& seg)
    {
      m_manner = seg.m_manner;
      m_voiced = seg.m_voiced;  m_strident = seg.m_strident;
      m_ctype = seg.m_ctype;
      return *this;
    }

private:
  Manner m_manner;
  bool m_voiced;
  bool m_strident;
  string m_ctype, m_label;  // canonical type, label
};

typedef map<string, SegmentClass> MannerMap;

MannerMap *construct_manner_table()
{
  MannerMap *manner_map = new MannerMap;

  if (manner_map) {
    // silence
    (*manner_map)["h#"] =  SegmentClass(Closure,   false, false, "sil");
    (*manner_map)["pau"] = SegmentClass(Closure,   false, false, "sil");
    (*manner_map)["epi"] = SegmentClass(Closure,   false, false, "sil");

    // closures
    (*manner_map)["pcl"] = SegmentClass(Closure,   false, false, "");
    (*manner_map)["bcl"] = SegmentClass(Closure,   true,  false, "");
    (*manner_map)["tcl"] = SegmentClass(Closure,   false, false, "");
    (*manner_map)["dcl"] = SegmentClass(Closure,   true,  false, "");
    (*manner_map)["kcl"] = SegmentClass(Closure,   false, false, "");
    (*manner_map)["gcl"] = SegmentClass(Closure,   true,  false, "");

    // stops
    (*manner_map)["p"] =   SegmentClass(StopBurst, false, false, "");
    (*manner_map)["b"] =   SegmentClass(StopBurst, true,  false, "");
    (*manner_map)["t"] =   SegmentClass(StopBurst, false, false, "");
    (*manner_map)["d"] =   SegmentClass(StopBurst, true,  false, "");
    (*manner_map)["k"] =   SegmentClass(StopBurst, false, false, "");
    (*manner_map)["g"] =   SegmentClass(StopBurst, true,  false, "");

    (*manner_map)["q"] =   SegmentClass(StopBurst, true,  false, "glott");
```

```
// fricatives
(*manner_map)["f"] =   SegmentClass(Fricative, false, false, "");
(*manner_map)["v"] =   SegmentClass(Fricative, true,  false, "");
(*manner_map)["th"] =  SegmentClass(Fricative, false, false, "");
(*manner_map)["dh"] =  SegmentClass(Fricative, true,  false, "");
(*manner_map)["s"] =   SegmentClass(Fricative, false, true,  "");
(*manner_map)["z"] =   SegmentClass(Fricative, true,  true,  "");
(*manner_map)["sh"] =  SegmentClass(Fricative, false, true,  "");
(*manner_map)["zh"] =  SegmentClass(Fricative, true,  true,  "");

(*manner_map)["ch"] =  SegmentClass(Fricative, false, true,  "afr");
(*manner_map)["jh"] =  SegmentClass(Fricative, true,  true,  "afr");

(*manner_map)["hh"] =  SegmentClass(Fricative, false, false, "h");
(*manner_map)["hv"] =  SegmentClass(Fricative, true,  false, "h");
(*manner_map)["ax-h"] = SegmentClass(Fricative, true, false, "ax-h");

// sonorant consonants and glides
(*manner_map)["m"] =   SegmentClass(Sonorant,  true,  false, "nasal");
(*manner_map)["n"] =   SegmentClass(Sonorant,  true,  false, "nasal");
(*manner_map)["ng"] =  SegmentClass(Sonorant,  true,  false, "nasal");
(*manner_map)["nx"] =  SegmentClass(Sonorant,  true,  false, "nasal");

(*manner_map)["l"] =   SegmentClass(Sonorant,  true,  false, "l");
(*manner_map)["r"] =   SegmentClass(Sonorant,  true,  false, "r");

(*manner_map)["w"] =   SegmentClass(Sonorant,  true,  false, "glide");
(*manner_map)["y"] =   SegmentClass(Sonorant,  true,  false, "glide");

(*manner_map)["dx"] =  SegmentClass(Sonorant,  true,  false, "flap");

// vowels
(*manner_map)["iy"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ih"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["eh"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ey"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ae"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["aa"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["aw"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ay"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ah"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ao"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["oy"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ow"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["uh"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["uw"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ux"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ax"] =  SegmentClass(Vocalic,   true,  false, "");
(*manner_map)["ix"] =  SegmentClass(Vocalic,   true,  false, "");
```

```
    (*manner_map)["er"] =  SegmentClass(Vocalic,   true,  false, "");
    (*manner_map)["axr"] = SegmentClass(Vocalic,   true,  false, "");

    (*manner_map)["el"] =  SegmentClass(Vocalic,   true,  false, "");

    (*manner_map)["em"] =  SegmentClass(Vocalic,   true,  false, "");
    (*manner_map)["en"] =  SegmentClass(Vocalic,   true,  false, "");
    (*manner_map)["eng"] = SegmentClass(Vocalic,   true,  false, "");

  }
  return manner_map;
}


void convert(ostream& os, istream& is, MannerMap* manner_map)
{
  // read in .phn label file:
  // for each pair of labels, characterize transition -> output expected LMs
  //   unfortunate issues:  - not all LMs will be realized
  //                          - unclear ordering in some cases

  TIMIT_label label, prev_label;
  SegmentClass curr_class, prev_class;
  bool ev[NUM_EV_TYPES], req[NUM_EV_TYPES], any_ev;

  is >> prev_label;       // grab first label, since we want transitions
  prev_class = (*manner_map)[prev_label.phon];

  while ( (is >> label).good() ) {
    // clear events
    for (int i = 0; i < NUM_EV_TYPES; i++) ev[i] = req[i] = false;

    curr_class = (*manner_map)[label.phon];

    // determine transition type(s)

    // a) is there a voicing boundary?  (+/- v)
    if ( !prev_class.is_sonorant() && curr_class.is_sonorant() )
      ev[ev_Von] = req[ev_Von] = true;
    else if ( prev_class.is_sonorant() && !curr_class.is_sonorant() )
      ev[ev_Voff] = req[ev_Voff] = true;

    // b) is there a sonorant consonantal boundary?  (+/- s)
    if ( prev_class.manner() == Sonorant && curr_class.manner() == Vocalic )
      ev[ev_Son] = req[ev_Son] = true;
    else if (prev_class.manner() == Vocalic && curr_class.manner() == Sonorant)
      ev[ev_Soff] = req[ev_Soff] = true;
    else if (prev_class.manner() == Sonorant &&
             curr_class.manner() == Sonorant)
      ev[ev_Son] = ev[ev_Soff] = true;
```

```
// c) is there an obstruent consonantal boundary? (+/-c)
if ( !prev_class.is_obstruent() && curr_class.is_obstruent() ) {
  ev[ev_Con] = true;
  req[ev_Con] = true;
}
else if ( prev_class.is_obstruent() && !curr_class.is_obstruent() ) {
  ev[ev_Coff] = true;
  if ( prev_class.manner() == Fricative ) req[ev_Coff] = true;
}
else if ( prev_class.is_obstruent() && curr_class.is_obstruent() ) {
  if ( prev_class.strident() && !curr_class.strident() )
    ev[ev_Coff] = true;
  else if ( !prev_class.strident() && curr_class.strident() )
    ev[ev_Con] = true;
  else
    ev[ev_Con] = ev[ev_Coff] = true;
}

// d) finally, make sure we output an event for an h#- or -h# boundary
if ( prev_class.manner() == Closure && !prev_class.is_canonical() ) {
  if ( !(ev[ev_Con] | ev[ev_Von]) ) ev[ev_Con] = true;
} else if ( curr_class.manner() == Closure && !curr_class.is_canonical() )
  if ( !(ev[ev_Coff] | ev[ev_Voff]) ) ev[ev_Coff] = true;

// output any events we've found
any_ev = false;
for (int i = 0; i < NUM_EV_TYPES; i++)
  if (ev[i]) {
    if (!any_ev) {
      os << label.start_time()*1000 << '\t'
         << prev_label.phon << ';' << label.phon << '\t';
      any_ev = true;
    }
    else os << ',';

    os << (event_type)i;
    if (!req[i]) os << '?';
  }
os << endl;


// e) extra +c for velar stop bursts (not req), and
// f) extra -c for (unvoiced?) stop bursts (assume asp. -- not req)
if ( curr_class.manner() == StopBurst && curr_class.is_canonical()
     /*&& !curr_class.voiced()*/ )
  write_LM(os, prev_label, label,
           ( label.phon == "k" || label.phon == "g" ) ? "+c?,-c?" : "-c?",
           "(burst off)", ( label.start_time() + label.end_time() )/2 );
```

```
    // ready for next iter
    prev_label = label;
    prev_class = curr_class;
  }
}


int main(int argc, char *argv[])
{
  int arg = 1;

  // check for other options
  if (argc-arg < 1 || argc-arg > 2) {
    cerr << "USAGE: phn2lm <phn file> [<output file>]"
         << endl
         << " <phn file> is the input label file" << endl
         << " <output file> defaults to stdout" << endl << endl;

    return -1;
  }

  // open input file
  ifstream inf( argv[arg] );
  if (!inf) {
    cerr << " Can't open input file '" << argv[arg] << "'." << endl;
    exit(-2);
  }

  // open output file
  ofstream outf( argv[++arg] );
  if (!inf) {
    cerr << " Can't open output file '" << argv[arg] << "'." << endl;
    exit(-2);
  }

  // set up manner definitions
  MannerMap *manner_map = construct_manner_table();

  // do translation
  convert(outf, inf, manner_map);

  inf.close();
  outf.close();
}
```

*C.2 DP Scoring (*compare *program)*

Main program: compare.cc

```
//
// compare.cc: compare and align recognized landmarks with expected
//   landmark sequence -- driver code
//

#include <fstream>
#include <list>

#include "compare.h"


//
// implementations

int main(int argc, char *argv[])
{
  int arg = 1;

  // check for other options
  if (argc-arg != 2) {
    cerr << "USAGE: compare <ref LM file> <recog LM file>" << endl << endl;
    return -1;
  }

  // open input file 1 -- ref
  ifstream inf1( argv[arg] );
  if (!inf1) {
    cerr << " Can't open input file '" << argv[arg] << "'." << endl;
    exit(-2);
  }

  // open input file 2 -- recog
  ifstream inf2( argv[arg+1] );
  if (!inf2) {
    cerr << " Can't open input file '" << argv[arg+1] << "'." << endl;
    exit(-2);
  }

  // read in each LM file
  LM_ref_label_seq labels_ref;
  LM_ref_label rlabel;
  LM_label_seq labels_recog;
  LM_label label;

  // get all of the labels
```

```
  while( (inf1 >> rlabel).good() )
    labels_ref.push_back(rlabel);
  while( (inf2 >> label).good() )
    labels_recog.push_back(label);

  // run alignment
  static penalty_table pt;
  pt.create_table(labels_ref, labels_recog);
  pt.backtrace(labels_ref, labels_recog);
  pt.output(cout, labels_ref, labels_recog);
  pt.output_results(cout);

  inf1.close();
  inf2.close();
}
```

Alignment DP algorithm: `align.cc`

```
//
// align.cc: compare and align recognized landmarks with expected
//   landmark sequence -- actual alignment code
//

#include <cmath>

#include "compare.h"

#define SUB_2X
static int dbug2 = 0;


ostream& operator << (ostream& os, const penalty_table::node &n)
{
  os << n.cost << ' ' << (int)n.p_step << ' '
     << (n.avail[0] ? '1':'-') << (n.avail[1] ? '2':'-');

  return os;
}


void penalty_table::create_table(LM_ref_label_seq& ref, LM_label_seq& recog)
{
  int i, j, cx, ct;

  n1 = ref.size();
  n2 = recog.size();

  the_table[0][0].fill(0, OK, ref[0]);
```

```
  cx = the_table[0][0].navail();
  for (i=1; i <= n1; i++) {
    the_table[i][0].fill(DEL_penalty * cx, DEL, ref[i]);
    cx += the_table[i][0].nreq(ref[i]);
  }
  for (j=1; j <= n2; j++) the_table[0][j].fill(INS_penalty * j, INS, ref[0]);

  for (i=1; i <= n1; i++) {
    for (j=1; j <= n2; j++) {

      cx = MAX_penalty;
      ct = the_table[i-1][j].cost
        + the_table[i-1][j].nreq(ref[i-1]) * DEL_penalty;
      if (ct<cx) cx=ct, the_table[i][j].fill(ct, DEL, ref[i]);

      /* try for matches from prev ref label: */
      if (the_table[i-1][j-1].navail() == 1) {
        event_type ev = ref[i-1].ev[ the_table[i-1][j-1].avail[0] ? 0:1 ];
        if (ev == recog[j-1].ev) {
          ct = the_table[i-1][j-1].cost
                + (int) (abs(ref[i-1].time-recog[j-1].time));
          if (ct<cx) cx=ct, the_table[i][j].fill(ct, OK, ref[i]);
        }
        else {
          ct = the_table[i-1][j-1].cost + SUB_penalty
#ifdef SUB_2X
               *(  // halved penalty if polarities match
                 ( polarity(ref[i-1].ev[ the_table[i-1][j-1].avail[0] ? 0:1 ])
                   == polarity(recog[j-1].ev) ) ? 1 : 2 )
#endif
               + (int) (abs(ref[i-1].time-recog[j-1].time));
          if (ct<cx) cx=ct, the_table[i][j].fill(ct, SUB, ref[i]);
        }
      }
      /* try for matches from current ref label: */
      if (the_table[i][j-1].navail() == 2) {
        if (ref[i].ev[0] == recog[j-1].ev) {
          ct = the_table[i][j-1].cost
                  + (int) (abs(ref[i].time-recog[j-1].time));
          if (ct<cx) cx=ct, the_table[i][j].fill(ct, OK2, ref[i], 0);
        } else if (ref[i].ev[1] == recog[j-1].ev) {
          ct = the_table[i][j-1].cost
                  + (int) (abs(ref[i].time-recog[j-1].time));
          if (ct<cx) cx=ct, the_table[i][j].fill(ct, OK2, ref[i], 1);
        } /* else {  // Don't allow subst for 2-avail cases ..
          ct= the_table[i-1][j-1].cost + SUB_penalty;
          if (ct<cx) cx=ct, the_table[i][j].cost=ct, the_table[i][j].p_step=SUB;
          } */
      }
```

```
      /* finally, insertion penalty case */
      ct = the_table[i][j-1].cost + INS_penalty;
      if (ct<cx) cx=ct, the_table[i][j].fill(ct, INS, the_table[i][j-1]);
    }
  }

  if (dbug2)
    for (i=0; i<=n1; i++) {
      for (j=0; j<=n2; j++)
      cout << the_table[i][j] << "   ";
      cout << endl;
    }
}


int penalty_table::backtrace(LM_ref_label_seq& ref, LM_label_seq& recog)
{
  int i, j, k, nn;
  float ubeg = ref.begin()->time, uend = (ref.end()-1)->time;
  //cout << "utt_beg: " << utt_beg << "\tutt_end: " << utt_end << endl;

  nerr = nsub = nins = ndel = nokdel = nokins = 0;

  for (k=0, i=n1, j=n2; i>=0 && j>=0; k++) {
    backtr[k] = the_table[i][j].p_step;
    switch (backtr[k]) {
    case OK:  i--; j--;                 break;
    case OK2:      j--;                 break;
    case SUB: i--; j--; nerr++; nsub++; break;
    case INS: if (recog[j-1].time < ubeg || recog[j-1].time > uend) nokins++;
                  j--; nerr++; nins++; break;
    case DEL: i--;  nn = the_table[i][j].navail();      // mult del possible
                    nokdel += nn - the_table[i][j].nreq(ref[i]);
                    nerr+=nn; ndel+=nn; break;

    default:  cerr << "backtrace: INVALID" << endl;
    }
  }

  if (dbug2)
    for (i = 0; i < k; i++)
      cout << "backtr[" << i << "] = " << (int)backtr[i] << endl;

  nbacktr = (--k);    /* return size after dumping faked entry */
  return nbacktr;
}


ostream& penalty_table::output(ostream& os,
                        LM_ref_label_seq& ref, LM_label_seq& recog)
```

```
{
  int i,j,m;
  LM_ref_label_seq::iterator i_ref = ref.begin();
  LM_label_seq::iterator i_recog = recog.begin();
  string s;

  for (m = nbacktr-1, i = j = 0; m >= 0; m--) {

    switch (backtr[m]) {
    case OK:
      os << *i_recog++ << '\t' << i_ref->ev[ the_table[i][j].avail[0] ? 0:1 ];
      if (!i_ref->req[ the_table[i][j].avail[0] ? 0:1 ]) os << '?';
      os << '\t' << i_ref->comment << endl;
      i++, j++, i_ref++;
      break;

    case OK2:
      os << *i_recog++ << '\t' << i_ref->ev[ the_table[i][j+1].avail[0] ? 1:0];
      if (!i_ref->req[ the_table[i][j+1].avail[0] ? 1:0 ]) os << '?';
      os << '\t' << i_ref->comment << endl;
      j++;
      break;

    case SUB:
      os << *i_recog++ << '\t' << i_ref->ev[ the_table[i][j].avail[0] ? 0:1 ];
      if (!i_ref->req[ the_table[i][j].avail[0] ? 0:1 ]) os << '?';
      os << '\t' << i_ref->comment << endl;
      i_ref++;
      i++, j++;
      break;

    case INS:
      // find transition label
      if (i_recog->time < ref.begin()->time)
      s = ";h#";   // assumption that initial segment label will be h#
      else
      s = (i_ref < ref.end() && i_recog->time >= i_ref->time)
            ? i_ref->comment : (i_ref-1)->comment;

      os << *i_recog++ << '\t' << '*' << '\t' << s.substr(s.find(';')+1) << endl;
      j++;
      break;

    case DEL:  // this is a mess, because either 1 or 2 evs can be del'd..
      os << i_ref->time << "\t*\t*\t";
      if (the_table[i][j].avail[0]) {
        os << i_ref->ev[0];  if (!i_ref->req[0]) os << '?';
      }
      if (the_table[i][j].navail() == 2) os << ',';
      if (the_table[i][j].avail[1]) {
```

```
        os << i_ref->ev[1];  if (!i_ref->req[1]) os << '?';
      }
      os << '\t' << (i_ref++)->comment << endl;
      i++;
      break;

    default:
      os << "????" << endl;
    }
  }

  os << endl;
  return os;
}

ostream& penalty_table::output_results(ostream& os)
{
  os << "Total errors:   " << nerr << endl
     << "Substitutions:  " << nsub << endl
     << "Insertions:     " << nins << endl
     << "Deletions:      " << ndel << endl
     << endl
     << "Accounted for:" << endl
     << " pre/post ins:  " << nokins << endl
     << " non-req dels:  " << nokdel << endl
     << "Net errors:     " << (nerr - nokins - nokdel) << endl;
  return os;
}
```

BIBLIOGRAPHY

[1] Kenneth N. Stevens. *Acoustic Phonetics*. MIT Press, Cambridge, Mass., 1999.

[2] George A. Miller and Patricia E. Nicely. An analysis of perceptual confusions among some english consonants. *Journal of the Acoustical Society of America*, 27(2):338–353, 1955.

[3] Dianne J. Van Tassell, Sigfrid D. Soli, Virginia M. Kirby, and Gregory P. Widin. Speech waveform envelope cues for consonant recognition. *Journal of the Acoustical Society of America*, 82(4):1152–1161, 1987.

[4] Dianne J. Van Tassell, Donna G. Greenfield, Joelle J. Logemann, and David A. Nelson. Temporal cues for consonant recognition: Training, talker generalization, and use in evaluation of cochlear implants. *Journal of the Acoustical Society of America*, 92(3):1247–1257, 1992.

[5] Robert V. Shannon, Fan-Gang Zeng, Vivek Kamath, John Wygonski, and Michael Ekelid. Speech recognition with primarily temporal cues. *Science*, 270:303–304, 1995.

[6] C. W. Turner, P. E. Souza, and L. N. Forget. Use of temporal envelope cues in speech recognition by normal and hearing-impaired listeners. *Journal of the Acoustical Society of America*, 97(4):2568–2576, 1995.

[7] P. B. Denes and E. N. Pinson. *The Speech Chain: The Physics and Biology of Spoken Language*. Anchor Press / Doubleday, Garden City, New York, 1973.

[8] S. J. Keyser and K. N. Stevens. Feature geometry and the vocal tract. *Phonology*, 11:207–236, 1994.

[9] Carol Y. Espy-Wilson. An acoustic-phonetic approach to speech recognition: Application to the semivowels. RLE Technical Report 531, MIT Research Lab for Electronics, June 1987.

[10] Carol Y. Espy-Wilson. A feature-based semi-vowel recognition system. *Journal of the Acoustical Society of America*, 96(1):65–72, 1994.

[11] Sharlene Liu. *Landmark Detection for Distinctive Feature-Based Speech Recognition*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1994.

[12] Sharlene Liu. Landmark detection for distinctive feature-based speech recognition. *Journal of the Acoustical Society of America*, 100(5):3417–3430, 1995.

[13] E. F. Evans. Auditory processing of complex sounds: an overview. In R. P. Carlyon, C. J. Darwin, and I. J. Russell, editors, *Processing of complex sounds by the auditory system*, pages 295–306. Oxford University Press, New York, 1992.

[14] E. F. Evans. Cortical representation. In A. V. S. de Reuck and J. Knight, editors, *Hearing representations in vertebrates*, pages 272–287. J. & A. Churchill, London, 1968.

[15] Philip X. Joris and Tom C. T. Yin. Envelope coding in the lateral superior olive. I. Sensitivity to interaural time differences. *Journal of Neurophysiology*, 73:1043–1062, 1995.

[16] Philip X. Joris. Envelope coding in the lateral superior olive. II. Characteristic delays and comparison with responses in the medial superior olive. *Journal of Neurophysiology*, 76:2137–2156, 1996.

[17] Philip X. Joris and Tom C. T. Yin. Envelope coding in the lateral superior olive. III. Comparison with afferent pathways. *Journal of Neurophysiology*, 79:253–269, 1998.

[18] Ariel Salomon and Carol Y. Espy-Wilson. Automatic detection of manner events based on temporal parameters. In *Eurospeech 99 Conference Proceedings*, volume 6, pages 2797–2800, 1999.

[19] Arindam Mandal, Laura J. Davis, Carol Y. Espy-Wilson, and Melanie Matthies. The use of spectral vs. temporal cues to recognize speech (2psc12). In *Program of the 138th Meeting of the Acoustical Society of America*, November 1999.

[20] Sangita Sharma and Hynek Hermansky. Speech recognition from temporal patterns. In *Proceedings, International Conference of Phonetic Sciences*, 1999.

[21] Nabil Bitar. *Acoustic Analysis and Modeling of Speech Based on Phonetic Features*. Ph.D. Dissertation, Boston University, Boston, 1997.

[22] Stuart Rosen. Temporal information in speech: acoustic, auditory, and linguistic aspects. *Philosophical transactions of the Royal Society of London Series B, Biological sciences*, 336:367–373, 1992.

[23] Brian C. J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, London / San Diego, 1999.

[24] S. P. Bacon and Neal F. Viemeister. Temporal modulation transfer functions in normal-hearing and hearing-impaired subjects. *Audiology*, 24:117–134, 1985.

[25] Neal F. Viemeister. Temporal modulation transfer functions based upon modulation thresholds. *Journal of the Acoustical Society of America*, 66(5):1364–1380, 1979.

[26] René van der Horst, A. Rens Leeuw, and Wouter A. Dreschler. Importance of temporal envelope cues in consonant recognition. *Journal of the Acoustical Society of America*, 105(3):1801–1809, 1999.

[27] Laurel H. Carney. A model for the responses of low-frequency auditory-nerve fibers in cat. *Journal of the Acoustical Society of America*, 93(1):401–417, 1993.

[28] Michael G. Heinz. *Quantifying the effects of the cochlear amplifier on temporal and average-rate information in the auditory nerve.* Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, Mass., 2000.

[29] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-time Signal Processing.* Prentice Hall, Englewood Cliffs, N.J., 1989.

[30] Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals.* Prentice Hall, Englewood Cliffs, N.J., 1978.

[31] Carol Y. Espy-Wilson. Acoustic measures for linguistic features distinguishing the semivowels /w j r l/ in american english. *Journal of the Acoustical Society of America*, 92(2):736–757, 1992.

[32] D. S. Pallett. Benchmark tests for DARPA resource management database performance evaluations (S10.b.6). In *Proceedings of the 1989 International Conference on Acoustics, Speech and Signal Processing*, pages 536–539, 1989.

[33] S. Seneff and V. Zue. Transcription and alignment of the TIMIT database. Included with the TIMIT database, 1988.

[34] J. Y. Choi, E. Chuang, D. Gow, K. Kwong, S. Shattuck-Hufnagel, K. N. Stevens, and Y. Zhang. Labeling a speech databse with landmarks and features (4asc6). In *Program of the 134th Meeting of the Acoustical Society of America*, page 3163, November 1997.

[35] ESPS 5.3.1. Entropic Research Laboratory, Inc., 1999.

[36] John F. Pitrelli. *Hierarchical Modeling of Phoneme Duration: Application to Speech Recognition*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1985.

VITA